

Java Coding Standards

James Yao and Andrew Okimi
May 30, 2006

Table of Contents

Design and Implementation	4
Overall Concepts of Design	5
Object Oriented Guideline	6
Java Design	9
Operations	10
Patterns, Terms and Notation.....	11
Appendix A.....	12
Naming operations	12
Appendix B	14
Glossary	14
Appendix C	21
Java Reference list	21

Design and Implementation

Introduction

It goes without saying that good design and implementation in development is very important.

Reusability, the component development process, etc. are all based on proper design and skilled implementation. It makes little difference if the component needs to be built for reuse, or for the current project only. Such a decision can be deferred to any phase or timing once the designer is following good Object Oriented Design (OOD) principles and the developer is coding well.

Design and development guidelines provide two high-level benefits: “they reinforce techniques to design and code better and they provide consistency even when there is no clear best choice.” [ChiMu]

Overall Concepts of Design

Knowing the standards

Industry standards are more desirable than organizational standards, which in turn are more desirable than project standards.

It is recommended that the Java standards, J2EE patterns, Object Oriented designs, etc. from Sun's website are consulted. Refer to Appendix for more information.

Understand the standards and follow them. Make them as a part of quality assurance process, e.g., design review.

Focus

Here are areas to focus on when thinking about project coding:

- Think from the client's point of view
 - Think from the maintainer's point of view
-

Suggestions

- **Once and only once.** A well-written program (with good style) will contain everything, once and only once.
 - **Lots of little pieces.** Good code has small methods and small objects.
 - **Replaceable components.** Good style leads to easily replaceable components.
 - **Pluggable components.** Indicates that components can be easily plugged to new contexts.
-

Object Oriented Guideline

Interface standards

[Interface with interfaces] An interface with interfaces is created differently for different types of clients. Most component classes should be implemented with component specification (interfaces).

Use interfaces as the glue that binds your code throughout, instead of classes. Define interfaces to describe the exterior of objects (i.e. their type) and type all variables, parameters, and return values to interfaces.

“Interface designers must take responsibility for the most important aspects of interface specification. Interfaces determine which aspects of a component are accessible and to whom they are accessible. Responsibility for interface usage errors belongs to the interface designer, not the interface user.” [MOST]

[Composition over inheritance] Object composition and inheritance are two techniques for reusing functionality in object-oriented systems.

Class inheritance, or sub classing, allows a subclass' implementation to be defined in terms of the parent class' implementation. This type of reuse is often called **white-box reuse**. The term refers to the fact that with inheritance, the parent class implementation is often visible to the subclasses.

Object composition is a different method of reusing functionality. Objects are composed to achieve more complex functionality. This approach requires that the objects have **well-defined interfaces** since the internals of the objects are unknown. Because objects are treated only as "black boxes", this type of reuse is often called **black-box reuse**.

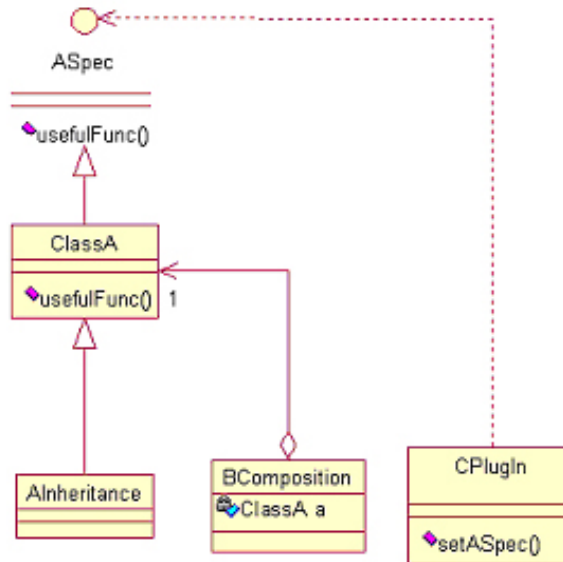
Note: Use *Thinking in Java* (see Appendix C) for more details.

Continued on next page

Object Oriented Guideline, Continued

Interface standards (continued)

In the following figure, CPlugin design is the best, then, Bcomposition design, the last choice is Ainheritance:



Guidelines

[Expose responsibility] (Name Types Well, Only Expose Responsibilities).

Spend the time to name a Type correctly and concisely. The name should match the range of usage for the Type: if it is very general, choose a very simple name; and if it is only applicable in a specific context, qualify it to describe that limitation.

Sometimes naming a Type can be very easy because it exists as part of the business concepts (an Employee) or part of the solution (a Window). But always make sure the name really matches the concept. This is very important.

Continued on next page

Object Oriented Guideline, Continued

Guidelines (continued)

Only expose the features that you want to be responsible for. A Type provides a contract to all its customers that it will need to maintain "for life", so never publicly expose something that someone does not want to be responsible for maintaining.

Make sure all instance variables and implementation specific methods are hidden from clients and do not become part of someone's responsibilities.

When following the above rule, if the name is not concise or the length of the name has to be very long, it is good practice to review the design. This can resolve if the design can be divided into a more concise shape.

Java Design

Guidelines

[Provision for the change] From a design standpoint, look for and separate things that change from things that stay the same.

[Move forward] Watch out for “analysis paralysis.” Remember that you must usually move forward in a project before you know everything, and that often the best and fastest way to learn about some of your unknown factors is to go to the next step rather than trying to figure it out in your head. You can’t know the solution until you have the solution. Java has built-in firewalls; let them work for you. Your mistakes in a class or set of classes won’t destroy the integrity of the whole system.[Think]

[Throw exception] Throw the runtime exception when the caller is not expecting such exception. E.g. throw SQLException is not good idea for a banker component. Try to throw an application exception instead, if it fits to the interface design. Or, such an exception can be designed as a runtime exception.

[Prefer protected to private] The method in the class is preferred as “protected” over “private”.

[Divide and conquer] Remember the “divide and conquer” principle. All software design problems can be simplified by introducing an extra level of conceptual indirection. If your code is too complicated, distribute the responsibility to smaller components or types.

Operations

Guidelines

[Write small methods that only do "one thing"] In particular, separate out methods that change an object state from those that just rely upon it. For a classic example in a Stack, prefer having two methods: **Object top()** and void **removeTop()** versus the single method **Object pop()** that does both.

Focus on communication and maintainability when implementing methods. "Divide your program into methods that perform one identifiable task. Keep all of the operations in a method at the same level of abstraction. This will naturally result in programs with many small methods, each a few lines long."
[ChiMu]

Patterns, Terms and Notation

Guidelines

[Common glossary] Having a common glossary of terms is important for accurate, precise, and concise communication among team members. See Appendix B.

[Patterns] Design patterns. J2EE patterns are very helpful, especially when they provide good vocabulary. Try to use these pattern names when a pattern is applied with proper modification. **Note:** In this document, we do not “encourage” the use of design pattern, as most Object Oriented students start with. The design patterns should be applied according to the application context and problem domain.

[UML notation] UML is recommended.

Appendix A

Naming operations

Prefixes

The following table provides examples of the naming convention using prefixes. The table also indicated their category, and adds descriptions.

Prefixes	Category	Description
General Prefixes		
is, can, has, will	Testing	Return a Boolean and test the state of the object
new	Creating	Create and return a new object from a factory that creates only a single type of object
init, setup	Initializing	These methods are called before you can use an object. Only a single init function should be called which can then be followed by whatever setup methods you need to change the default configuration of the object.
Type Specific Prefixes		
Find	Searching	Retrieve a single object or null if unsuccessful
select	Searching	Retrieve multiple objects or an empty collection
add	N/A	Add an object to a collection

Continued on next page

Naming operations, Continued

Non-prefix The following table provides examples of non-prefix naming convention

Operation Name	Category	Description
Any	N/A	Return any object that satisfies the request (findAny)
All	N/A	Return all objects that satisfy the request (selectAll)

Categories for method

The following list provides categories for method:

- **Constructing** a section and category. The constructors for the class.
- **Initializing** an additional method that should be applied directly after constructing the object.
- **Setup Methods** that can optionally be applied to an object but must be done immediately after construction and initialization and before using the object normally.
- **Validating Check** whether the current object is in an acceptable state (could also be under asking if this is possible after construction is finishing).
- **Asking.** Asking the state of the current object without causing any (visible) side effects. A pure function. ISE Eiffel 'Query'.
- **Testing.** An asking method that returns a Boolean value.

Appendix B

Glossary

Terms

The following definitions are ChiMu's distilling and reconciliation of the many concepts and work that have been contributed to Object Oriented Design. Many of the sources for these terms are mentioned in this document. Some other notable sources are:

- The Dictionary of Object Technology [Firesmith+E 95].
- UML: The Unified Modeling Language [Rational 98]
- Design Patterns, especially [Gamma+HJV 96]

Term	Description
Adapter	An object that can convert an Interface of one Class to the interface that another Object expects
Architecture	A concepts, structures, and interactions of a system. Also, the description of a system's desired architecture before construction.
Association ₁	A relationship between two Types that allow or require Objects of those Types to be linked.
Association ₂	An Association ₁ , but must be between Types which have Objects with Identity. See Attribute and ValueObject.
Attribute1	A public property of an object that shows an aspect of the state of the object. Frequently there is a minimal collection of attributes that uniquely determine the state of the object. See also Property.
Attribute2	See BasicAttribute.
Attribute3	See Instance Variable.
BasicAttribute	An Attribute1 that takes its value from ValueObjects. This is as opposed to associations, which connect two or more objects with identity. A BasicAttribute is traversable only from the Object to the ValueObject.
Bean	An Object that knows about its own properties (can introspect) and has several other capabilities. Any Java Object can be a Bean but some Objects have more "Bean" functionality.

Continued on next page

Glossary, Continued

Terms (continued)

Term	Description
Behavior	The response of an object to a stimulus. An object's behavior is the answers it gives to messages both now and in the future. (See State).
Binding	Associating a client object to a database object, which turns the client object into a Proxy
Class	Provides a common implementation for a set of objects.
Container	A Registry but without the implication of a primary registration property (e.g. a "key").
DomainModel	All the static rules, constraints, and operations that apply to DomainObjects. The DomainModel can either be conceptual to help understand the behavior of DomainObjects or it can be implemented as DomainClasses
DomainObject	An object, which captures knowledge about a domain. DomainObjects allow a computer to inspect, imply, modify, and "reason" about that information in either very simple ways (the facts) or more complex ways (the rules and implications)
Extend	To define a new Type (called a Subtype) in terms of an existing type (called a Supertype). The new Subtype will have the same contract (operations) as the Supertype but can add new functionality: as either new operations or enhancement in capability to existing operations.
Extent	The collection of all instances of a Type or Class.
ExtentRegistry	A Registry that contains all the objects of a Type (the Extent of the Type). An ExtentRegistry is a close equivalent to a RelVar or Table in the context of Objects.
Factory	An object that can create other objects.
Feature	The Eiffel term for Operation where Operation includes both methods and attributes ₁ .
Forwarder	A proxy, which immediately forwards messages, over process and machine boundaries, to the RealSubject.

Continued on next page

Glossary, Continued

Terms (continued)

Term	Description
Framework	A strong partition of generalized functionality common to many parts of an application. Also, more formally, a collection of interacting classes that describes most of the behavior a client requires and can be subclassed and parameterized to customize and complete the functionality.
Function _f	A functor that returns an Object
Functor	An object that models an operation.
Functor	“An object that models an operation” [Firesmith+E 95]. For a Java implementation, a basic functor is an Interface with a single, generic, operation.
Getter _f	A functor that is designed to retrieve a value from an object (the first parameter)
Identity	The ability to tell one object from another object independently of whether their appearance (behavior) is identical
IdentityKey	A value that defines the RealIdentity of a Proxy
Immutable	Can not be changed after being created. Immutable objects can not be changed after they are created and fully initialized.
Inherit	To define a new Class in terms of an Existing Class (the Superclass) by starting with the Superclass’s implementation and i-directi or adding to it
Instance	An object is an Instance of a Type if an object supports all the exterior requirements of that type (see “Is-A”). An object is an instance of a Class if it is implemented by that class
Instance Variable	A way to store encapsulated state information for a particular object. Instance variables are completely hidden within the Object, but they enable two objects of the same Class to have different external Behavior.
Interface	A description of a Type focused on the Operations that the objects can respond to.

Continued on next page

Glossary, Continued

Terms (continued)

Term	Description
Is-A	An object is a Type if an Object supports all the exterior requirements of that Type
Layer	A logical, horizontal division of a system that provides a particular system abstraction to the client above the layer.
Link	A connection between two objects which allows one or both to know about the other object. By default links are assumed to be i-directional in analysis, but they can be defined to be only traversable in one direction
Message	A stimulus sent to an object with a name and any parameters (as Objects) that the message requires. A message will cause the receiver Object to return an answer or nothing. In most Object Languages, the sender has to wait for the answer before continuing
Method	An implementation of an operation for a particular object/class.
Module	A base level subsystem: one which does not contain any other subsystems
Object	An identifiable, encapsulated entity that can only be interacted with by sending messages
ObjectBase	An ObjectBase captures the knowledge, operations, and rules required to usefully represent a particular part of the world in a computer. An ObjectBase contains all the objects that represent a particular state of your DomainModel and all the knowledge contained therein. Also called an ObjectSpace
ObjectShadow	The information needed to see that an object exists without any true representation of the real object. Relational databases could be considered to work with ObjectShadows: they record the information about an object but never have a real object to interact with.

Continued on next page

Glossary, Continued

Terms (continued)

Term	Description
Observable	An object that can be Observed. See Observer
Observer	An object that “looks at” another object (the Observable) and can respond to events in the Observable without the Observable being knowledgeable about the Observer
Operation	A description of the ability for an object to respond to a particular message and the contract/requirement for that message.
Partition	A vertical division of a system into areas of related functionality
Predicate _f	A functor that returns a Boolean.
Procedure _f	A functor that does not return a value.
Property	Synonym for Attribute ₁ and sometimes for Attribute ₂
Prototype	An object that is used as a template for creating other Objects
Proxy	An object that stands in for another object (the RealObject) and manages the client interaction with the RealObject
RealIdentity	The identity of the RealObject that a proxy represents instead of the proxy’s independent identity. For proxies we are rarely interested in their own identity, we just want to know the identity of the RealObject on the server.
Registry	An object that remembers other objects and can search through and retrieve them through one or more properties. Usually the objects within a Registry are all of the same type
Replicate	A proxy which holds local state and performs local operations which are later propagated to the RealSubject
Role	The name of the “position” within a relationship an Object or Type holds. For example, a binary association has two roles that distinguish the two participants in the relationship

Continued on next page

Glossary, Continued

Terms (continued)

Term	Description
Setter _f	A functor that stores into an object (the first parameter) a value (the second parameter)
Singleton	An object that is the only instance of a Class
State	An abstraction to describe and simplify understanding an object's behavior. An object's behavior can be described as the answers it gives to current messages (which are determined by the current state) and the changes to its state caused by these messages
Strategy	An object that encapsulates an algorithm to be used with an Object
Stub	A proxy which acts as a placeholder for the RealObject and must become another type of proxy (for example, forwarder or replicate) when interacted with by a client
Subsystem	A division of a system into a cohesive unit of functionality (tightly related classes and internal subsystems) with a public interface and a private implementation
Subtype	A Type that extends another Type (called the Supertype)
Supertype	A Type that has been extended by another Type
Tier	A level on a hierarchy of processes over which a system is divided
Traverse	To move from one object to another by a Link. If a Link is traversable from an Object than that Object can get to (knows about) the other Object
Type	Describes a common exterior (public behavior) of a set of Objects. Can also be used to conceptually group and understand objects by their similar behavior

Continued on next page

Glossary, Continued

Terms (continued)

Term	Description
ValueObject	An object that does not have identity independent of its value. A ValueObject is immutable and should be considered identical to anything that it is equal to. Primitive data types in Smalltalk (most numbers, Symbols) are ValueObjects. Java Strings are very close to ValueObjects except they are not guaranteed to be identical for the same value (they would be if they did an automatic “intern()”). Java primitive types are not Objects.
Visitor	An object representing an operation that can be performed on the elements of an Object structure (frequently a hierarchy or sequence).

Appendix C

Java Reference list

List The following is a list of manufacturers and their URLs where more information can be found on various Java coding standards and guidelines.

Manufacturer	URL
ChiMu	http://www.chimu.com/publications/javaStandards/index.html
The AmbySoft Inc.	http://www.ambysoft.com/downloads/javaCodingStandards.pdf
MOST	"The Most Important Design Guideline" by Scott Meyer
IEEE	http://www.aristeia.com/Papers/IEEE_Software_JulAug_2004.pdf
SUN Microsystems	http://java.sun.com/j2se/javadoc/writingdoccomments/index.html
SUN Microsystems	http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
Hammurapi	Open source software developed by Pavel Vlasov, Hammurapi is a tool that can analyze a code base against a set of design guidelines. There is a plug in available for Eclipse (WSAD). See: http://www.hammurapi.org
Bruce Eckel	<i>Thinking in Java</i> is a book written by Bruce Eckel. For more information, go to: http://www.BruceEckel.com
