JAVA NAMING CONVENTIONS

A **good** developer knows that there is more to development than programming.
A **great** developer knows that there is more to development than development.

When you go against a standard, document it. All standards, except for this one, can be broken. You must document why you broke the standard, the potential implications of breaking the standard, and any conditions that may/must occur before the standard can be applied to this situation.

Always use (a few exceptions discussed below) full English descriptors. Use lower case letters in general, but capitalize the first letter of class / interface names and the first letter of any non-initial word

General Concepts

Use terminology applicable to the domain
Use mixed case for readability
Use short forms sparingly and intelligently
Avoid long names (< 15 characters)
Avoid names that are similar or differ only in case

Item	Example	Naming convention
Arguments/ parameters	customer , account , - or - aCustomer , anAccount	Full English description of value/object being passed, possibly prefixing the name with ‘a’ or ‘an.’
Fields / properties	firstName , lastName , warpSpeed	Full English description, 1st letter lowercase, 1st letter of any non-initial word in uppercase
Boolean getter member functions	isPersistent() , isString() , isCharacter()	Pefixed with ‘is’
Classes	Customer , SavingsAccount	Full English description, with the first letters of all words capitalized
Compilation unit files	SavingsAccount.java , Singleton.java	Name of class/interface; if > 1 class in file, prefixed with ‘.java’ to indicate it's a source code file.
Components / widgets	okButton , customerList , fileMenu	Full English description describing usage; type of the component concatenated onto the end.
Contructors	Customer() , SavingsAccount()	Use the name of the class
Destructors	finalize()	Will invoke finalize() member function before an object is garbage collected
Exceptions	e	It is generally accepted to use the letter ‘e’ to represent exceptions
Final Static fields / constants	MIN_BALANCE , DEFAULT_DATE	Uppercase letters, words separated by under-scores. Better: final static getter member functions
Getter member functions	getFirstName() , getWarpSpeed()	Prefix the name of the field being accessed with ‘get’
Interfaces	Runnable , Prompter , Singleton	Full English descr. concept of interface, 1st letters of words cap'd. *Postfix name with ‘able,’ ‘ible,’ or ‘er’*
Local variables	grandTotal , customer , newAccount	Full English description, 1st letter in lower case but do not hide existing fields/fields
Loop counters	i , j , k , counter	It is generally accepted to use the letters i , j , or k , or the name ‘counter .’
Package	ca.uvic.neptune.per-sistence.mapping	See Classes - Global packages: reverse name of Internet domain & postfix the package name.
Member Functions	openFile() , addAccount()	Full English description of what it does; starting with active verb if possible, 1st letter in lower case
Setter member functions	setLastName() , setWarpSpeed()	Prefix the name of the field being accessed with ‘set’

JAVA DOCUMENTATION CONVENTIONS

Rule of thumb: if you've never seen the code before, what documentation would you need to quickly understand it

General Concepts

Comments should add to clarity
If it isn't worth documenting, it isn't worth running
No decoration / banner-like comments
Keep comments **simple**
Write documentation **before** writing code
Why ~ not **What**

Java comment types

Documentation	Immediately before declarations of interfaces, classes, member functions and fields to document them. These are processed by javadoc to create external documentation for a class.	`/** Customer – A customer is any person or organization that we sell services and products to. @author S.W. Ambler */`
C Style	C-style comments to disable lines of code that are no longer applicable, but that you want to keep just in case ~ or while debugging.	`/* Commented out by J.T. Kirk on 1/1/03 replaced by preceding code. Delete after 2 years if still not applicable . . . (the source code ) */`
Single line	Use single line comments internally within member functions to document business logic, code sections and declarations of temporary variables.	`// Apply a 5% discount to all // invoices over $1000 as defined by // the Sarek generosity campaign // started in Feb 1995`

WHAT to document

Arguments / parameters	The type of the parameter What it should be used for Any restrictions or preconditions Examples
Fields/properties	Its description Document all applicable invariants Examples Concurrency issues Visibility decisions
Classes	The purpose of the class Known bugs The development/maintenance history of the class Document applicable invariants The concurrency strategy
Compilation units	Each class/interface defined in the class, incl. a brief description The file name and/or identifying information Copyright information
Getter member function	Document why lazy initialization was used, if applicable
Interfaces	The purpose How it should and shouldn't be used
Local variables	Its use/purpose
Member Functions (Documentation)	What and why the member function does what it does What a member function must be passed as parameters What a member function returns Known bugs Any exceptions that a member function throws Visibility decisions How a member function changes the object Include a history of any code changes Examples of how to invoke the member function if appropriate Applicable preconditions and postconditions Document all concurrency
Member Functions (Internal comments)	Control structures Why, as well as what, the code does Local variables Difficult or complex code The processing order
Package	The rationale for the package The classes in the package

JAVA CODING CONVENTIONS

99.9% of the time it is more important to program **for your fellow developers** than for the machine

Your code must be understandable to others

Accessor member functions	Consider using lazy initialization for fields in the database Use accessors for obtaining and modifying all fields Use accessors for 'constants" For collections, add member functions to insert and remove items Whenever possible, make accessors protected, not public
Fields	Fields should always be declared private Do not directly access fields, instead use accessor member functions Do not use final static fields (constants), instead use accessor member functions Do not hide names Always initialize static fields
Classes	Minimize the public and protected interfaces Define the public interface for a class before you begin coding it Declare the fields and member functions of a class in the following order: · constructors · finalize() · public member functions · protected member functions · private member functions · private field
Local variables	Do not hide names Declare one local variable per line of code Document local variables with an endline comment Declare local variables immediately before their use Use local variables for one thing only
Member functions	Document your code Paragraph your code Use whitespace, one line before control structures and two before member function declarations A member function should be understandable in less than thirty seconds Write short, single command lines Restrict the visibility of a member function as much as possible Specify the order of operations

This text is a summary of Scott Ambler's 'Writing Robust Java Code'

The AmbySoft Inc. Coding Standards for Java - v17.01d

This layout by Maike Dulk