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.

### Java Naming Conventions

**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

<table>
<thead>
<tr>
<th>Item</th>
<th>Example</th>
<th>Naming convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments/parameters</td>
<td>customer, account, - or</td>
<td>Full English description of value/object being passed, possibly prefixing the name with 'a' or 'an.'</td>
</tr>
<tr>
<td>Fields / properties</td>
<td>firstName, lastName,</td>
<td>Full English description, 1st letter lowercase, 1st letter of any non-initial word in uppercase</td>
</tr>
<tr>
<td></td>
<td>warpSpeed</td>
<td></td>
</tr>
<tr>
<td>Boolean getter member functions</td>
<td>isPersistent(), isString(),</td>
<td>Pefixed with 'is'</td>
</tr>
<tr>
<td></td>
<td>isCharacter()</td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>Customer,</td>
<td>Full English description, with the first letters of all words capitalized</td>
</tr>
<tr>
<td></td>
<td>SavingsAccount</td>
<td></td>
</tr>
<tr>
<td>Compilation unit files</td>
<td>SavingsAccount.java,</td>
<td>Name of class/interface; if &gt; 1 class in file, prefixed with '.java' to indicate it's a source code file.</td>
</tr>
<tr>
<td></td>
<td>Singleton.java</td>
<td></td>
</tr>
<tr>
<td>Components / widgets</td>
<td>okButton, customerList,</td>
<td>Full English description describing usage; type of the component concatenated onto the end.</td>
</tr>
<tr>
<td></td>
<td>fileMenu</td>
<td></td>
</tr>
<tr>
<td>Constructors</td>
<td>Customer(),</td>
<td>Use the name of the class</td>
</tr>
<tr>
<td></td>
<td>SavingsAccount()</td>
<td></td>
</tr>
<tr>
<td>Destructors</td>
<td>finalize()</td>
<td>Will invoke finalize() function before an object is garbage collected</td>
</tr>
<tr>
<td>Exceptions</td>
<td>e</td>
<td>It is generally accepted to use the letter 'e' to represent exceptions</td>
</tr>
<tr>
<td>Final Static fields / constants</td>
<td>MIN_BALANCE,</td>
<td>Uppercase letters, words separated by underscores. Better: final static getter member functions</td>
</tr>
<tr>
<td></td>
<td>DEFAULT_DATE</td>
<td></td>
</tr>
<tr>
<td>Getter member functions</td>
<td>getFirstName(),</td>
<td>Prefix the name of the field being accessed with 'get'</td>
</tr>
<tr>
<td></td>
<td>getWarpSpeed()</td>
<td></td>
</tr>
<tr>
<td>Interfaces</td>
<td>Runnable, Prompter,</td>
<td>Full English descr. concept of interface, 1st letters of words cap'd. Postfix name with 'able', 'ible', or 'er'</td>
</tr>
<tr>
<td></td>
<td>Singleton</td>
<td></td>
</tr>
<tr>
<td>Local variables</td>
<td>grandTotal, customer,</td>
<td>Full English description, 1st letter in lower case but do not hide existing fields/fields</td>
</tr>
<tr>
<td></td>
<td>newAccount</td>
<td></td>
</tr>
<tr>
<td>Loop counters</td>
<td>i, j, k, counter</td>
<td>It is generally accepted to use the letters i, j, or k, or the name 'counter.'</td>
</tr>
<tr>
<td>Package</td>
<td>ca.uvic.neptunepersistence.mapping</td>
<td>See Classes - Global packages; reverse name of Internet domain &amp; postfix the package name.</td>
</tr>
<tr>
<td>Member Functions</td>
<td>openFile(), addAccount()</td>
<td>Full English description of what it does; starting with active verb if possible, 1st letter in lower case</td>
</tr>
<tr>
<td>Setter member functions</td>
<td>setLastName(),</td>
<td>Prefix the name of the field being accessed with 'set'</td>
</tr>
<tr>
<td></td>
<td>setWarpSpeed()</td>
<td></td>
</tr>
</tbody>
</table>
Rule of thumb: if you’ve never seen the code before, what documentation would you need to quickly understand it?

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.
- **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.
- **Single line**: Use single line comments internally within member functions to document business logic, code sections and declarations of temporary variables.

Arguments / parameters

- The type of the parameter
- What it should be used for
- Any restrictions or preconditions

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

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

- The purpose
- Document why lazy initialization was used, if applicable
- How it should and shouldn’t be used

Local variables

- Its use/purpose
- 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

Member Functions (Documentation)

- 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

**WHAT to document**

Accessors/properties

- 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