The purpose of the [47 Degrees](http://47deg.com/) Java Coding Standards is to create a collaboration baseline; helpful for scenarios where many people are creating, modifying, and contributing to the same project. The end goal is unity, consistency, and the notion that a single entity worked on the project.

Java applications developed at *47 Degrees* should follow the [Code Conventions for the Java Programming Language](http://www.oracle.com/technetwork/java/codeconv-138413.html)

*47 Degrees* uses the Java Programming Language and Java Related Technologies in different scenarios and applications. As a norm *47 Degrees* follows industry standards and notifying of any deviation of this document from these standards will be welcomed.

**Table of Contents** *generated with*[*DocToc*](http://doctoc.herokuapp.com/)

* [Java Coding Standards](https://github.com/47deg/coding-guidelines/tree/master/java#java-coding-standards)
  + [0. Rebel](https://github.com/47deg/coding-guidelines/tree/master/java#0-rebel)
  + [1. General](https://github.com/47deg/coding-guidelines/tree/master/java#1-general)
  + [2. Comments](https://github.com/47deg/coding-guidelines/tree/master/java#2-comments)
  + [3. Copyrights](https://github.com/47deg/coding-guidelines/tree/master/java#3-copyrights)
    - [3.1. Open Source](https://github.com/47deg/coding-guidelines/tree/master/java#31-open-source)
    - [3.2. Propietary - Not Open Source](https://github.com/47deg/coding-guidelines/tree/master/java#32-propietary---not-open-source)
    - [3.3. 3rd Parties](https://github.com/47deg/coding-guidelines/tree/master/java#33-3rd-parties)
  + [4. Good Practices](https://github.com/47deg/coding-guidelines/tree/master/java#4-good-practices)
    - [4.1. Design for interfaces](https://github.com/47deg/coding-guidelines/tree/master/java#41-design-for-interfaces)
    - [4.2. Define Application Layers.](https://github.com/47deg/coding-guidelines/tree/master/java#42-define-application-layers)
    - [4.3. Avoid multiple return statements](https://github.com/47deg/coding-guidelines/tree/master/java#43-avoid-multiple-return-statements)
    - [4.4. Boolean comparisons](https://github.com/47deg/coding-guidelines/tree/master/java#44-boolean-comparisons)
    - [4.5. for loops Vs for-each loops](https://github.com/47deg/coding-guidelines/tree/master/java#45-for-loops-vs-for-each-loops)
    - [4.5. String concatenation](https://github.com/47deg/coding-guidelines/tree/master/java#45-string-concatenation)
    - [4.6. Exceptions](https://github.com/47deg/coding-guidelines/tree/master/java#46-exceptions)
    - [4.7. Collections](https://github.com/47deg/coding-guidelines/tree/master/java#47-collections)
    - [4.8. Raw types](https://github.com/47deg/coding-guidelines/tree/master/java#48-raw-types)
    - [4.9. Use of final](https://github.com/47deg/coding-guidelines/tree/master/java#49-use-of-final)
    - [4.10. Name return values result](https://github.com/47deg/coding-guidelines/tree/master/java#410-name-return-values-result)
    - [4.11. Consider setters and getters for field access](https://github.com/47deg/coding-guidelines/tree/master/java#411-consider-setters-and-getters-for-field-access)
    - [4.12. Know your Java](https://github.com/47deg/coding-guidelines/tree/master/java#412-know-your-java)
  + [5. Design Patterns](https://github.com/47deg/coding-guidelines/tree/master/java#5-design-patterns)
    - [5.1. Abstract Factory](https://github.com/47deg/coding-guidelines/tree/master/java#51-abstract-factory)
    - [5.2. Factory method](https://github.com/47deg/coding-guidelines/tree/master/java#52-factory-method)
    - [5.3. Lazy Delegate Wrapper](https://github.com/47deg/coding-guidelines/tree/master/java#53-lazy-delegate-wrapper)
    - [5.4. Singleton](https://github.com/47deg/coding-guidelines/tree/master/java#54-singleton)
    - [5.5. Enums](https://github.com/47deg/coding-guidelines/tree/master/java#55-enums)
    - [5.6. Private Helpers](https://github.com/47deg/coding-guidelines/tree/master/java#56-private-helpers)
  + [6. Other Java Technologies](https://github.com/47deg/coding-guidelines/tree/master/java#6-other-java-technologies)
  + [7. Active enforcement](https://github.com/47deg/coding-guidelines/tree/master/java#7-active-enforcement)

**0. Rebel**

Do NOT blindly obey these guidelines, use them where they make sense.

**1. General**

* All code, regardless of who touches it, should look like a single person created it.
* All culprits who ignore the aforementioned rule will be docked one pint of beer.

**2. Comments**

Well documented Java code is as awesome as a [pizzokie](http://www.urbandictionary.com/define.php?term=pizzokie). We have all fallen victim to undocumented code; even being guilty of ignoring proper comments for our own code. Let's try and stop doing that.

At *47 Degrees* comments in Java files should be written as [Doc Comments for the Javadoc Tool](http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html).

**Correct:**

/\*\*

\* Returns a user object that can be stored in the data store.

\* The name argument must be alphanumeric.

\* <p>

\* This method always creates a new user, whether or not a user

\* with the same name already exists.

\*

\* @param name the name given to a new user

\* @return the newly created user

\* @see OtherClassOfInterest

\*/

public User createUser(String name) {

User user = new User();

user.setName(name);

return user;

}

**Incorrect:**

// creates a user

public User createUser(String name) {

User user = new User(); // a new user is always created

user.setName(name);

return user;

}

**3. Copyrights**

All Java files should contain the appropiate copyright notice at the beginning of the file.

**3.1. Open Source**

Open Source code written in *47 Degrees* that is open source is commonly released under the [Apache License, Version 2.0](http://www.apache.org/licenses/LICENSE-2.0.html) and must be as follows.

/\*

\* Copyright (C) {year} 47 Degrees, LLC

\* http://47deg.com

\* hello@47deg.com

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

**3.2. Propietary - Not Open Source**

Code owned and written for 47 Degrees not for open sourced projects must be as follows.

/\*

\* Copyright (C) {year} 47 Degrees, LLC

\* http://47deg.com

\* hello@47deg.com

\* All rights reserved

\*/

**3.3. 3rd Parties**

Code owned and written for 3rd parties should be formatted in the following way.

/\*

\* Copyright (C) {year} {3rdPartyName}

\* {3rdPartUrl}

\* All rights reserved

\* Developed for {3rdPartyName} by:

\*

\* 47 Degrees, LLC

\* http://47deg.com

\* hello@47deg.com

\*/

**4. Good Practices**

Good practice are a must at *47 Degrees*. Code is periodically reviewed for both machines and humans to ensure code quality meets standards within the company.

Pay special attention to the most common style errors...

* classes too long
* methods too long
* little or no javadoc comments
* swallow exceptions
* multiple *return* statements
* Overuse of arrays in place of collections
* too much or no whitespace

In the spirit of providing a guideline to avoid some of these errors and create better code the below listed points are cases with examples that should be considered at 47 Degrees.

**4.1. Design for interfaces**

**Correct:**

/\*\*

\* Defines the contract for the user related operations

\* @see User

\*/

public interface UserService {

}

/\*\*

\* Implements the UserService connecting and saving the user to the database

\* @see UserService

\*/

public class LocalUserServiceImpl implements UserService {

}

**Incorrect:**

/\*\*

\* Does not implements the UserService but connects and saves the user to the database

\* @see UserService

\*/

public class UserService {

}

**Correct:**

public List<User> getFriends(User user) {

}

**Incorrect:**

public ArrayList<User> getFriends(User user) {

}

**4.2. Define Application Layers.**

**Correct:**

/\*\*

\* User interface or API response (Serialized to JSON or used in view technologies such as JSP)

\*/

public class UserResponse {

private String name;

public UserResponse(User user) {

setName(user.getName());

}

public String getName() {

return this.name;

}

public void setName(String name) {

this.name = name;

}

}

/\*\*

\* Handles UI interaction and communication to services (Servlets, Spring MVC Controllers, Android activities, Tapestry pages...)

\*/

public class UserController {

private UserService userService;

public UserResponse onCreateUserFormSubmit(String name) {

User user = userService.createUser(name);

return new UserResponse(user);

}

}

/\*\*

\* Implements business logic (Spring Bean Services)

\*/

public interface UserService {

User createUser(String name);

}

/\*\*

\* Implements the UserService

\* @see UserService

\*/

public class LocalUserServiceImpl implements UserService {

private PersistenceAdapter persistenceAdapter;

public User createUser(String name) {

User user = new User();

user.setName(name);

persistenceAdapter.save(user);

return user;

}

}

/\*\*

\* Entity that gets persisted to the data store

\*/

public class User {

private String name;

public String getName() {

return this.name;

}

public void setName(String name) {

this.name = name;

}

}

/\*\*

\* Handles persistence (DAO, Cassandra Persistence Adapters, JPA Persistence Adapters)

\*/

public interface PersistenceAdapter {

void save(Object object);

}

**Incorrect:**

/\*\*

\* Donwloads and sorts the interwebs, all in one method

\*/

public class MagicClass {

void doMagic(Object... args) {

//5000 lines here

}

}

**4.3. Avoid multiple *return* statements**

Multiple *return* statements are hard and time consuming to debug.

**Correct:**

public class StringUtils {

public static boolean isEmpty(String string) {

return string == null || "".equals(string.trim());

}

}

or

public class StringUtils {

public static boolean isEmpty(String string) {

boolean empty = false;

if (string == null) {

empty = true;

} else if ("".equals(string.trim())) {

empty = true;

}

return empty;

}

}

**Incorrect:**

public class StringUtils {

public static boolean isEmpty(String string) {

if (string == null) {

return true;

} else if ("".equals(string.trim())) {

return true;

}

return false;

}

}

**4.4. Boolean comparisons**

Mirroring the natural language "if the current state is not active" rather than "if active is not the current state"

**Correct:**

!active

**Incorrect:**

active == false

**4.5. *for* loops Vs *for-each* loops**

When itereating over iterable elements where the current index in the iteration is not important for-each loops are preffered.

**Correct:**

for (String name: names) {

doSomething(name);

}

**Incorrect:**

for (int i = 0; i < names.length; i++) {

doSomething(names[i]);

}

**4.5. *String* concatenation**

Avoid the use of + or += to concatenate strings. Use java standards designed for that purposes such as String.format, StringBuilder, etc.

**Correct:**

log.debug(String.format("found %s items", amount));

**Incorrect:**

log.debug("found " + amount + " items");

**4.6. Exceptions**

Don't swallow exception, catch those you can recover or do something about, let the rest reach their destination

**Correct:**

try {

do();

} catch (SomethingWeCanHandleException ex) {

log.error(ex);

notifyUser(ex);

} finally {

cleanUp();

}

public void doSomething() throws SomethingSomeoneElseCanHandleException {

}

**Incorrect:**

try {

do();

} catch (Throwable ex) {

log.error(ex);

}

try {

do();

} catch (SomethingWeCanHandleException ex) {

//do nothing

}

**4.7. Collections**

Use the right collections for the right task.

**Duplicates**

* Allows duplicates: [List](http://docs.oracle.com/javase/6/docs/api/java/util/List.html)
* Does Not Allow Duplicates: [Set](http://docs.oracle.com/javase/6/docs/api/java/util/Set.html), [Map](http://docs.oracle.com/javase/6/docs/api/java/util/Map.html)

**Implementations Iteration Order**

* [HashSet](http://docs.oracle.com/javase/6/docs/api/java/util/HashSet.html) - undefined
* [HashMap](http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html) - undefined
* [LinkedHashSet](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedHashSet.html) - insertion order
* [LinkedHashMap](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedHashMap.html) - insertion order of keys
* [ArrayList](http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) - insertion order
* [LinkedList](http://docs.oracle.com/javase/6/docs/api/java/util/LinkedList.html) - insertion order
* [TreeSet](http://docs.oracle.com/javase/6/docs/api/java/util/TreeSet.html) - ascending order (Comparable / Comparator)

**4.8. Raw types**

Avoid using raw types when using classes that support generics.

**Correct:**

List<String> people = Arrays.asList("you", "me");

**Incorrect:**

List people = Arrays.asList("you", "me");

**4.9. Use of *final***

When implementing services and classes that are more than javabeans or objects to transfer data between layers make sure you use the 'final' keyword to communicate your intention regarding subclassing, use of constants and values that once set should be immutable.

public final class ThisShouldNeverBeExtended {

...

}

public final neverOverrideThisMethod() {

}

private final int thisFielfValueWillNeverChangeOnceSet;

final int thisLocalVariableValueWillNeverChangeOnceSet = 0;

**4.10. Name return values *result***

Consider using 'result' as the name for the returned variable. This eases the pain when debugging and increases code legibility.

public Object doSomething() {

Object result = null;

if (something) {

result = new Object();

}

return result;

}

**4.11. Consider setters and getters for field access**

Within the same class consider using getters and setter to access fields values to ensure lazy initialization and other intended logic implemented in getters and setters is always applied.

**Correct:**

public void changeName(String name) {

setName(name);

}

**Incorrect:**

public void changeName(String name) {

this.name = name;

}

**4.12. Know your Java**

Spend some time getting to know the [core packages](http://docs.oracle.com/javase/6/docs/api/overview-summary.html) that are already part of the SDK and avoid reinventing the wheel.

**5. Design Patterns**

Consider the use of common design patterns.

**5.1. Abstract Factory**

Creates or gets an object without knowledge of its implementations, minimizing the refactoring effort and keeping implementations well defined and isolated into their classes.

public interface PersistenceAdapter {

...

}

public interface CassandraPersistenceAdapter extends PersistenceAdapter {

...

}

public interface JPAPersistenceAdapter extends PersistenceAdapter {

...

}

public class HibernateJPAPersistenceAdapterImpl implements JPAPersistenceAdapter {

...

}

public class HectorCassandraPersistenceAdapterImpl implements CassandraPersistenceAdapter {

...

}

//Obtains the current runtime impl for the persistence adapter.

PersistenceAdapter persistenceAdapter = getPersistenceAdapter();

persistenceAdapter.save(user);

**5.2. Factory method**

Use static factory methods in objects where you may not create a new instance every time or you can return a subtype of the declared return type

Calendar.getInstance();

**5.3. Lazy Delegate Wrapper**

When coding classes that represent response objects utilized in Views or other parts of the system where a subset of the properties may not be used consider the use of lazy wrappers with delegates keeping in mind the Serialization requirements.

public class UserResponse {

private User delegate;

public UserResponse(User delegate) {

this.delegate = delegate;

}

public String getName() {

return delegate.getName();

}

}

**5.4. Singleton**

Use a singleton to represent a service class where there should be a single instance in the whole system. There is usually no need for implementing this pattern manually in web and backends where instances are managed by Spring or other IOC container.

public class Earth {

private static Earth instance = new Earth();

private Earth() {

}

public static Earth getInstance() {

return instance;

}

}

**5.5. Enums**

Constrain arguments by using type safe enumerations.

**Correct:**

public enum Options {

YES, NO

}

**Incorrect:**

String yes = "YES";

String no = "NO";

**5.6. Private Helpers**

Consider private helper methods to break down complex flows and long methods into more readable code.

**Correct:**

public void downloadUrlContents(String url) {

checkIfHostIsReachable(url);

saveContentsToFile(url);

}

**Incorrect:**

public void downloadUrlContents(String url) {

... complex code to see if a host is reachable

... complex code to turn the remote response into bytes and then serialize to disk

}

**6. Other Java Technologies**

Other Java related technologies along with their specific coding guidelines covered as subsets of this document can be found at:

* [Spring](https://github.com/47deg/coding-guidelines/blob/master/java/java/spring)
* [Android](https://github.com/47deg/coding-guidelines/blob/master/java/java/android)

**7. Active enforcement**

Java developers at *47 Degrees* are expected to use tools that enforce some of these good practices and quality code in general. The de facto standard at 47 degrees for coding Java applications is the IDE [IntelliJ IDEA](http://www.jetbrains.com/idea/). Developers that want to further improve their code and have many of the guidelines in this document auto enforced may use [QAPlug](http://qaplug.com/) enabled as plugin within IDEA. In order to organize methods based on their visibility and name we use [Rearranger](http://plugins.intellij.net/plugin/?id=173). The latest company template to rearrange methods in classes is available for download in this project's [Download](https://github.com/47deg/coding-guidelines/downloads) section

*47 Degrees* unapologetically favores the use of [IntelliJ IDEA](http://www.jetbrains.com/idea/) and other [products in the Jetbrains family](http://www.jetbrains.com/products.html) as standard tools for coding Java, ObjectiveC and other programming languages utilized at 47 Degrees