**Java Naming Conventions**

These are guidelines that Java Software Developers should follow when naming identifiers to produce consistent and readable code throughout the application. If teams do not follow these conventions, they may collectively write an application code that is hard to read and difficult to understand.

Java heavily uses [**CamelCase**](https://en.wikipedia.org/wiki/Camel_case) notations for naming the methods, variables, etc. and [**TitleCase**](https://en.wikipedia.org/wiki/Letter_case) notations for classes and interfaces.

The following are the fundamental rules that every identifier must follow:

* The name must not contain any white spaces.
* The name should not start with special characters like & (ampersand), $ (dollar), \_ (underscore).

Let's see some other rules that identifiers should follow:

1. **Packages naming conventions:**

The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, like *com, edu, gov, mil, net, org*.

Example:

package org.springframework.core.convert;

package org.hibernate.criterion;

package org.springframework.boot.actuate.audit;

package org.apache.tools.ant.dispatch;

Package naming convention used by Oracle for the Java core packages. The initial package name representing the domain name must be in lower case.

package java.lang;

package java.util;

1. **Classes naming conventions:**

Class names should be *nouns* in *UpperCamelCase*(in mixed case with the first letter of each internal word capitalized). Try to keep your class names simple and descriptive.

Example:

class Employee

class Student

class EmployeeDao

class CompanyService

The class naming convention used by Oracle for the Java core packages:

class String

class Color

class Button

class System

class Thread

class Character

class Compiler

class Number

1. **Interfaces naming conventions:**

In Java, interface names, generally, should be **adjectives**. Interfaces should be in title case with the first letter of each separate word capitalized. In some cases, interfaces can also be nouns when they present a family of classes, e.g. *List* and *Map*.

Example:

Runnable

Remote

ActionListener

Appendable

AutoCloseable

CharSequence

Cloneable

Comparable

Readable

1. **Methods naming conventions:**

Methods should be **verbs**. They represent action, and the method name should clearly state the action. The method name can be single or 2-3 words to convey the action. Words should be in **camel notation**.

Examples:

public List <Customer> getCustomers();

public void saveCustomer(Customer theCustomer);

public Customer getCustomer(int theId);

public void deleteCustomer(int theId);

More Examples:

getName()

computeTotalWidth()

actionPerformed()

main()

print()

println(),

1. **Variables naming conventions:**

The variable name should start with a lowercase letter. We should write parameter names, member variable names, and local variable names in *lowerCamelCase*.

Example:

firstName

orderNumber

lastName

phoneNo

id

counter

temp

1. **Constants naming conventions:**

We should write constant variable names in upper characters separated by *underscores*. These names should be semantically complete and precise.

Example:

RED, YELLOW, MAX\_PRIORITY, MAX\_STOCK\_COUNT

1. **Abstract Classes naming conventions:**

In many standard libraries, I observed that the naming convention used for *Abstract* class is that the class name must start with *Abstract or Base* prefix. This naming convention can vary from organization to organization.

Example:

AbstractHibernateDao

AbstractCommonDao

AbstractBase

Let's take an example from the Spring Framework:

AbstractBean

AbstractBeanDefinition

AbstractUrlBasedView

AbstractIdentifiable

1. **Exception Classes naming conventions:**

In many standard libraries, the naming convention used for custom *Exception* classes is the class name must end with *Exception* suffix.

Example:

TransactionException

SQLDataException

ResourceNotFountException

ResourceAlreadyExistException

Oracle uses the exception class naming convention for the Java core packages.

ArithmeticException

ArrayIndexOutOfBoundsException

ArrayIndexOutOfBoundsException

ClassNotFoundException

CloneNotSupportedException

EnumConstantNotPresentException

Exception

IllegalAccessException

IllegalArgumentException

IllegalMonitorStateException

IllegalStateException

IllegalThreadStateException

IndexOutOfBoundsException

1. **Enumeration naming conventions:**

Enum Class members should be spelled out in upper case words, separated by underlines. Example:

public enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

THURSDAY, FRIDAY, SATURDAY;

}

1. **Enumeration naming conventions:**

Generic type parameter names should be single uppercase letters. For example, the letter *'T'* for a type is typically recommended. In JDK classes, *E* is used for collection elements, *S* is used for service loaders, and *K* and *V* are used for map keys and values.

public interface Map <K,V> {}

public interface List<E> extends Collection<E> {}

Iterator<E> iterator() {}

1. **Annotations naming conventions:**

Annotation names follow title case notation. They can be adjectives, verbs or nouns based on the requirements.

public @interface FunctionalInterface {}

public @interface Deprecated {}

public @interface Documented {}

public @Async Documented {

public @Test Documented {

1. **Specific naming conventions:**

Apart from the above java standard naming conventions, we would follow a few more naming conventions in many standard libraries such as Spring, Apache, Hibernate, etc.

Note that these naming conventions can vary as per different libraries or organizations.

***is* prefix can be used for boolean variables and methods.**

isSet, isVisible, isFinished, isFound, isOpen

That is the naming convention for *boolean* methods and variables used by Oracle for the Java core packages.

Using the is prefix solves a common problem of choosing bad boolean names like status or flag. *isStatus* or *isFlag* simply doesn't fit, and the programmer is forced to choose more meaningful names.

Setter methods for boolean variables must have set prefixes as in:

void setFound(boolean isFound);

There are a few alternatives to the *is* a prefix that fits better in some situations. These have, can and should prefixes:

boolean hasLicense();

boolean canEvaluate();

boolean shouldAbort = false;

**The term *compute* can be used in methods where something is calculated.**

Example:

valueSet.computeAverage();

matrix.computeInverse()

Give the reader an immediate clue that this is a potentially time-consuming operation, and if used repeatedly, he might consider caching the result. Consistent use of the term enhances readability.

**The term *find* can be used in methods where something is looked up.**

vertex.findNearestVertex();

matrix.findSmallestElement();

node.findShortestPath(Node destinationNode);

Give the reader an immediate clue that this is a simple lookup method with a minimum of computations involved. In addition, consistent use of the term enhances readability.

**The term *initialize* can be used where an object or a concept is established.**

printer.initializeFontSet();

**We should use the *Plural* form on names representing a collection of objects.**

Collection<Point> points;

int[] values;

Enhances readability since the name gives the user an immediate clue of the type of the variable and the operations that can be performed on its elements.

**Generic variables should have the same name as their type.**

Example:

void setTopic(Topic topic) // NOT: void setTopic(Topic value)

// NOT: void setTopic(Topic aTopic)

// NOT: void setTopic(Topic t)

void connect(Database database) // NOT: void connect(Database db)

// NOT: void connect(Database oracleDB)

**The object's name is implicit and should be avoided in a method name.**

Example:

line.getLength(); // NOT: line.getLineLength();

// End