# Java/J2EE Naming Convention

**[07/04/2010]**

# Table of Contents

# Java/J2EE Naming Convention

## 1. Introduction

This document describes the naming standards, conventions, and guidelines for writing solid Java code. They are based on sound, proven software engineering principles that lead to code that is easy to understand, to maintain, and to enhance.

## 2. Purpose

The purpose of this document is to define the naming convention for Java/J2EE components of EGovernment Portal application.

## 3. Scope

The scope of this document is to define the naming convention for basic Java artifacts and Java/J2EE components in the EGovernment Portal application.

## 4. Definitions, Acronyms, and Abbreviations

None

## 5. References

Please refer *Portal Application Framework*.

## 6. Overview

This document details the naming conventions for the basic Java artifacts. It also details the naming conventions for Java/J2EE components in the Portal Application Framework.

# 7. Naming Convention

Defining the naming convention is vital because they lead to greater consistency within your code and the code of your teammates. Greater consistency leads to code that is easier to understand, which means it is easier to develop and to maintain. Thus, it reduces the overall cost of the applications that you create.

You have to remember that your Java code will exist for a long time even after you have moved on to other projects. An important goal during development is to ensure that you can transition your work to another developer or to another team of developers so they can continue to maintain and enhance your work without having to invest an unreasonable effort to understand your code. Code that is difficult to understand runs the risk of being scrapped and rewritten.

# 8. Naming Convention for Basic Java Components

Before get started, let us define the basic naming convention principles.

Use full English descriptors that accurately describe the variable, field, and class; for example, use names like firstName, grandTotal, or CorporateCustomer. Although names like x1, y1, or fn are easy to type because they're short, they do not provide any indication of what they represent and result in code that is difficult to understand, maintain, and enhance.

Use terminology applicable to the domain. If your users refer to their clients as customers, then use the term Customer for the class, not Client. Many developers make the mistake of creating generic terms for concepts when perfectly good terms already exist in the industry or domain.

Use mixed case to make names readable. Use lowercase letters in general, but capitalize the first letter of class names and interface names, as well as the first letter of any non-initial word.

Use abbreviations sparingly, but if you do so then use them intelligently. This means you should maintain a list of standard short forms (abbreviations), you should choose them wisely, and you should use them consistently. For example, if you want to use a short form for the word "number", then choose one of nbr, no, or num, document which one you chose (it does not really matter which one), and use only that one.

Avoid long names (< 15 characters is a good idea). Although the class name PhysicalOrVirtualProductOrService might seem to be a good class name at the time, this name is simply too long and you should consider renaming it to something shorter, perhaps something like Offering.

Avoid names that are similar or differ only in case. For example, the variable names persistentObject and persistentObjects should not be used together, nor should anSqlDatabase and anSQLDatabase.

Avoid leading or trailing underscores. Names with leading or trailing underscores are usually reserved for system purposes and may not be used for any user-defined names except for preprocessor defines.

## 8.1 Naming Field

A field is a piece of data that describes an object or class. Fields may be base data type, like a string or a float, or may be an object, such as a customer or a bank account.

You should use a full English descriptor to name your fields, thereby making it obvious what the field represents. Fields that are collections, such as arrays or vectors, should be given names that are plural to indicate that they represent multiple values.

***Examples:***

```
firstName

zipCode

unitPrice

discountRate

orderItems
```

## 8.2 Naming Constant

In Java, constants— values that do not change—are typically implemented as static final fields of classes. The recognized convention is to use full English words, all in uppercase, with underscores between the words.

***Examples:***

```
MINIMUM_BALANCE

MAX_VALUE

DEFAULT_START_DATE
```

The main advantage to this convention is that it helps you distinguish constants from variables.

## 8.3 Naming Component

For names of components (interface widgets), you should use a full English descriptor postfixed by the widget type. This makes it easy for you to identify the purpose of the component, as well as its type, which makes it easier to find each component in a list. Many visual programming environments provide lists of all components in an applet or application and it can be confusing when everything is named button1, button2, and so on.

***Examples:***

```
okButton

customerList

fileMenu

newFileMenuItem
```

## 8.4 Naming Member Function

Member functions should be named using a full English description, using mixed case with the first letter of any non-initial word capitalized. It is also common practice for the first word of a member function name to be a strong, active verb.

***Examples:***

```
openAccount()

printMailingLabel()

save()

delete()
```

This convention results in member functions whose purpose can often be determined just by looking at their names. Although this convention results in a little extra typing by the developer because it often results in longer names, this is more than made up for by the increased understandability of your code.

## 8.5 Naming Getter and Setter

**Getters**
Getters are member functions that return the value of a field. You should prefix the word "get" to the name of the field, unless it is a boolean field and then you prefix "is" to the name of the field instead of "get".

***Examples:***

```
getFirstName()

getAccountNumber()

isPersistent()

isAtEnd()
```

By following this naming convention, you make it obvious that a member function returns a field of an object, and for boolean getters you make it obvious that it returns true or false. Another advantage of this standard is that it follows the naming conventions used by the Beans Development Kit (BDK) for getter member functions.

Alternative naming convention for Getters: has and can

A viable alternative, based on proper English conventions, is to use the prefix "has" or "can" instead of "is" for boolean getters. For example, getter names such as hasDependents() and canPrint() make a lot of sense when you are reading the code. The problem with this approach is that the BDK will not pick up on this naming strategy (yet). You could rename these member functions isBurdenedWithDependents() and isPrintable().

**Setters**
Setters, also known as mutators, are member functions that modify the values of a field. You should prefix the word "set" to the name of the field, regardless of the field type.

***Examples:***

```
setFirstName(String aName)

setAccountNumber(int anAccountNumber)

setReasonableGoals(Vector newGoals)

setPersistent(boolean isPersistent)

setAtEnd(boolean isAtEnd)
```

By following this naming convention, you make it obvious that a member function sets the value of a field of an object. Another advantage of this standard is that it follows the naming conventions used by the BDK for setter member functions.

## 8.6 Naming Constructor

Constructors are member functions that perform any necessary initialization when an object is first created. Constructors are always given the same name as their class. For example, a constructor for the class Customer would be Customer(). Note that the same case is used.

***Examples:***

```
Customer()

SavingsAccount()

PersistenceBroker()
```

This naming convention is set by Sun Microsystems and must be strictly adhered to.

## 8.7 Naming Collection

A collection, such as an array or a vector, should be given a pluralized name representing the types of objects stored by the array. The name should be a full English descriptor, with the first letter of all non-initial words capitalized.

***Examples:***

```
customers

orderItems

aliases
```

The main advantage of this convention is that it helps to distinguish fields that represent multiple values (collections) from those that represent single values (non-collections).

## 8.8 Common Guideline

When you cannot find a rule or guideline, you can apply a rule that best describes the component in the Java code artifact.

# 9. Naming Convention for Portal Application Framework Components

Having defined the naming conventions for the basic Java components, let us define the naming conventions for each component in the Portal Application Framework. When you name any Java/J2EE component, you need to follow all naming conventions mentioned in the Section 8 as well as additional naming rule, defined in the following sections, in order to distinguish various layers in the Portal Application Framework.

## 9.1 Naming Value Object

Value Objects are data holders for the Managed Beans and upper layers. Each Value Object must have its **toString** method overridden and should provide copy constructors to facilitate ease of copying. Value Objects should not contain any business logic.

The Value Objects should start with name of eService and end with the word VO in the uppercase. In case the eService name is too long, you can use Service Code or an abbreviation of the service name.

***Examples:***

For the Payment Request entity, the name of Value Object can be `PaymentRequestVO`.

For the Transaction Data entity of Formation of Regulation Case, the name of Value Object can be `RCTransactionDataVO`. In this case, the service code RC is used to indicate the service.

For the Applicant entity of Submit Citizen Letter, the name of Value Object can be `SCLApplicantVO`.

## 9.2 Naming Business Controller

This is the business logic layer of the model. This layer encapsulates business rule validations, calculations, data accesses, and other logic that drives the central functionality of Portal application. The components designed in this layer could be a plain Java or an EJB component.

The Business Controller should start with name of eService and end with the word Controller as it is. In case the eService name is too long, you can use Service Code or an abbreviation of the service name.

***Examples:***

For the Poll service, the name of the Business Controller can be `PollController`.

For the Regulation Case service, the name of the Business Controller can be `RCController`. In this case, the service code RC is used to indicate the service.

## 9.3 Naming Business Object

Business Objects are ordinary Java Beans which carry data downward from the Business Controller layer. All Business Objects must have **toString** method overridden and should provide copy constructors to facilitate ease of copying.

The Business Objects should start with name of eService and end with the word BO in the uppercase. In case the eService name is too long, you can use Service Code or an abbreviation of the service name.

***Examples:***

For the Calendar object, the name of the Business Object can be `CalendarBO`.

For the Case entity of Regulation Case service, the name of the Business Object can be `RCCaseBO`. In this case, the service code RC is used to indicate the service.

## 9.4 Naming Data Access Layer

This is a generic layer for accessing data. Data can be stored in any form such as Relational Databases, MQ, XML, Flat Files, etc. Data Access Layer encapsulates the functionality to work with these technologies and provides an abstract handle through which the upper layers can seamlessly interact.

The Data Access Layer should start with name of eService and end with the word DAO in the uppercase. In case the eService name is too long, you can use Service Code or an abbreviation of the service name.

***Examples:***

For the Poll Service, the name of Data Access Layer can be `PollDAO`.

For the Case entity of Regulation Case service, the name of the Data Access Layer can be `RCDAO`. In this case, the service code RC is used to indicate the service.