# Naming Conventions

## Class and Interface Names

Class names are always nouns, not verbs. Avoid making a noun out of a verb, example DividerClass. If you are having difficulty naming a class then perhaps it is a bad class.

Interface names should always be an adjective (wherever possible) describing the enforced behaviors of the class (noun). Preferably, said adjective should end in "able" following an emerging preference in Java. i.e. Clonable, Versionable, Taggable, etc.

Class, and interface names begin with an uppercase letter, and shoule not be pluralized unless it is a collection class (see below).

**Acceptable:**

```
class FoodItem
interface Digestable
```

**Unacceptable:**

```
class fooditem
class Crackers
interface Eat
```

Naming collection classes (in the generic sense of collection) can be tricky with respect to pluralization. In general each collection should be identified as a plural item, but not redundantly so. If you are a collection type as part of the class name (List, Map, etc.) it is not necessary to use the plural form in the class name. If you are not using the collection type in the name it is necessary to pluralize the name. If you are extending one of the java colletction class (Map, HashMap, List, ArrayList, Collection, etc.) it is good practice to use the name of the collection type in the class name.

**Acceptable:**

```
class FoodItems extends Object
class FoodItemList extends ArrayList
class FoodItemMap extends HashMap
```

**Unacceptable:**

```
class FoodItem extends ArrayList
class FoodItemsList extends ArrayList
```

Class names should be descriptive in nature without implying implementation. The internal implementation of an object should be encapsulated and not visible outside the object. Since implementation can change, to imply implementation in the name forces the class name and all references to it to change or else the code can become misleading.

**Acceptable:**

```
AbstractManagedPanel
LayeredPanel
```

**Unacceptable:**

```
PanelLayerArray
```

When using multiple words in a class name, the words should be concatenated with no separating characters between them. The first letter of each word should be capitalized.

**Acceptable:**

```
InverntoryItem
```

**Unacceptable:**

```
Inverntory_item
Inventoryitem
```

Other than prefixes, no abbreviations should be used unless it is a well known abbreviation.

**Acceptable:**

```
CD=Compact Disc
US=United States
```

**Unacceptable:**

```
Cust=Customer
DLR=Dealer
```

## Method Names

Method names are typically verbs. However they can also be nouns for example accessor methods (see accessor methods below). In general, when a method modifies the object (or a related object) somehow use a verb appropriate to the nature of the modification. i.e. set, free, sort etc. If the method is an accessor method use a noun appropriate to the information that is returned.

**Acceptable:**

```
label getTag()
void setWidth(int)
void resetCounter()
```

**Unacceptable:**

```
label tagIs()
void counterNew()
```

Names should reflect exactly what the method does (no more or no less). A method should only have a single purpose. If your method contains too much functionality, then you should break it into more than one method.

Strive for names that promote self documenting code. The method name should read well in the code. Picture how the method will appear in your code.

Method and function names begin with a lowercase letter. The initial letter of any subsequent words in the name are capitalized, and underscores are not used to separate words.

**Acceptable:**

```
setInitState()
getAttributeCollection()
```

**Unacceptable:**

```
set_init_state()
getattributecollection()
```

Method names should be defined so as to describe the function of the method without implying implementation.

**Acceptable:**

```
addItem()
getItem()
```

**Unacceptable:**

```
addItemToVector()
getHashItem()
```

## Attribute and Local Variable Names

Do not use abbreviations, use full names. Variable names begin with a lowercase letter. The initial letter of any subsequent words in the name are capitalized, and underscores are not used to separate words (or scope variables). Clarity of variable names can be increased by providing some indication of the type of class they might become. Attributes that are not collections should not be pluralized.

**Acceptable:**

```
Item menuItem;
JPanel managerJPanel;
```

**Unacceptable:**

```
JPanel Managerpanel;
JPanel Manager_panel;
int _localInt;
InventoryItem i;
```

Collection classes, such as vectors and hashes should always be pluralized. Alternately, collection classes can also be prefixed with the word some.

**Acceptable:**

```
Vector menuItems;
Vector menuItemsVector;
Vector someMenuItems
```

**Unacceptable:**

```
Vector menuItemVector;
Vector aBunchOfMenu
```

Name variables with the most abstract class that they can hold. For example if `startButton` could be any control object, then it should be named a `startControl` .

If the variable truly represents an anonymous object but is restricted by an interface, then including the interface name in the variable can increase clarity. i.e. `clonableInventoryItem` .

Declare each variable separately on a single line. Do not comma separate variables of the same type.

**Acceptable:**

```
int counter;
int lastCounter;
```

**Unacceptable:**

```
int counter, lastCounter;
```

Constant values should have uppercase letters for each word and each word should be separated by an underscore. The capitalization of constants helps to distinguish them from other nonfinal variables.

**Acceptable:**

```
public final static int MAX_AGE = 100;
public final static Color RED = new Color(count++);
```

**Unacceptable:**

```
public final static int MAXAGE = 100;
public final static int maxAge = 100;
public final static int MaxAge = 100;
```

## Returning Arrays and Lists

Any method that will returns an list of homogeneous or heterogeneous items should return a Collection (or other collection class) object -- never an array.

Example: for a method that returns a list of keys represented as strings.

**Acceptable:**

```
List getKeys();
```

**Unacceptable:**

```
String[] getKeys();
```

Also, any method that returns a Collection should always return a valid Collection -- never null. However, the returned Collection can be empty.

**Acceptable:**

```
public ArrayList getKeys() {
   if (0 == this.numValidKeys) {
      return new ArrayList();
   }
   return myKeyList;
}
```

**Unacceptable:**

```
public ArrayList getKeys() {
   if (0 == this.numValidKeys) {
      return null;
   }
   return myKeyList;
}
```

## Don't "Hide" Names

Name hiding refers to the practice of naming a local variable, argument, or field the same (or similar) as that of another of greater scope. For example, if you have a class attribute called `firstName` do not create a local variable called `firstName` or anything close to it, such as `firstNames` or `fName`. Try to avoid this, it makes you code difficult to understand and prone to bugs because other developers will misread your intentions and create difficult to detect errors. [1]

---

1. **http://www.iwombat.com/standards/JavaStyleGuide.html#Methods**