# How to Use Class Variables and Class Instance Variables

Juanita J. Ewing

Instantiations, Inc.



Derived from Smalltalk Report

In last month's column I discussed some strategies for initializing classes and how initialization related to class variables and class instance variables. In this column I will talk about coding conventions for class variables, and when to use class variables vs. class instance variables.

Classes that use class variables can be made more reusable with a few coding conventions. These coding conventions make it easier to create subclasses. Sometimes developers use class variables inappropriately. Inappropriate use of class variables results in classes that are difficult to subclass. Often, the better implementation choice for a particular problem is a class instance variable instead of a class variable.

**What are class variables?** Classes can have

- class variables, and
- class instances variables.

Class variables are referenced from instance and class methods by referring to the name of the class variable. Any method, either a class method or an instance method can reference a class variable. Figure 1 contains a diagram of a class, ListInterface, that defines a class variables.

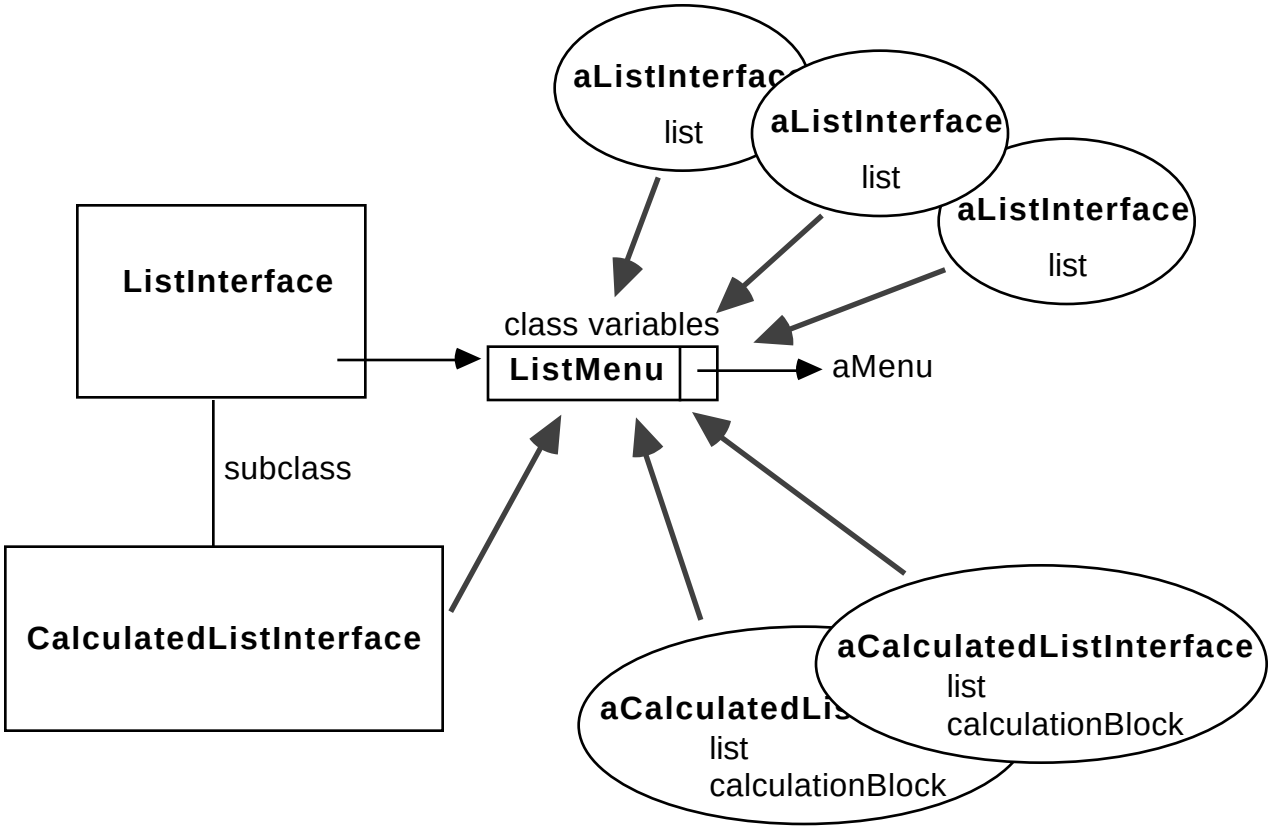


Figure 1. Class variables are referenced by subclasses and all instances.

The methods in ListInterface would look like this:

```
ListInterface class
    initialize
        "Create a menu. "
        ListMenu := Menu labels: #('add' 'remove')
ListInterface
    hasMenu
        "Return true if a menu is defined."
        ^ListMenu notNil
    performMenuActivity
        "Perform the mouse-based activity for my view."
        self hasMenu
            ifTrue: [^ListMenu startUp].
```

Both instance and class methods can directly reference class variables by name. The class method initialize is used to bind values to the class variables. The instance methods hasMenu and performMenuActivity reference the class variable ListMenu. All instances of ListInterface and the class ListInterface share the same class variables.

**How are class variables inherited?** Class variables and the values they are bound to are inherited. The class variable referenced by a subclass is the same as the one referenced

by the superclass. This means that a class variable is shared by a class, all its subclasses, and all the instances of the class and its subclasses.

Our example has a subclass of ListInterface, called CalculatedListInterface. Subclass methods referring to the ListMenu class variable reference exactly the same object as the superclass method. The subclass CalculatedListInterface has behavior that is different from its superclass, as defined by the method conditionalMenuActivity:

```
CalculatedListInterface
    conditionalMenuActivity
        "Perform the mouse-based activity for my view if the
list is not empty.
        If there is no menu, flash the list pane."
        self hasMenu
            ifFalse: [^self flash].
         list isEmpty
            ifFalse: [^ListMenu startUp].
```

Subclass methods can directly reference class variables that are defined by the superclass. In our example, the CalculatedListInterface method references the class variable ListMenu that is defined by ListInterface. This is different from the inheritance of instance variables. The method conditionalMenuActivity references the instance variable list that is defined by the class ListInterface. But, each instance of CalculatedListInterface and ListInterface has its own copy of list and does not share its instance variables.

**How do subclasses modify class variables?** It is possible for subclass methods to modify inherited class variables, but generally it is undesirable to do so. If a subclass were to modify a class variable, it would change the only existing value of the class variable. Each subclass does **not** have its own copy. It references a shared copy. Generally, developers want to create a new class variable and use it in place of the inherited class variable.

Using our example we will create a new menu in the subclass CalculatedListInterface. The menu is implemented with a class variable so it is not possible to change the menu for the subclass without also changing it for the superclass. This is because both classes reference the same variable.

The only way to create a new menu for the subclass and retain the original menu for the superclass is to create a new class variable. In our example we call the new class variable CalculatedListMenu. In addition to a new class variable, all methods that reference the original menu must be overridden in the subclass:

```
CalculatedListInterface class
    initialize
        "Create a calculated menu."
        CalculatedMenu :=  Menu labels: #('add' 'remove'
'print')
CalculatedListInterface
    hasMenu
        "Return true if a menu is defined."
        ^CalculatedMenu notNil
```

```
    performMenuActivity
        "Perform the mouse-based activity for my view."
        self hasMenu
            ifTrue: [^CalculatedMenu startUp].
```

Because direct references to the class variable ListMenu are sprinkled throughout the class ListInterface, the subclass must override many methods. In this simple example, we had to override three methods that reference ListMenu in order to reference a different menu. In a complicated real-world application, many other methods may need to be overridden in order to reference a different class variable in a subclass. Because significant portions of the class needed to be overridden, the class is not very reusable.

A better version of ListInterface has the minimum number of references to a class variable; one for setting and one for retrieving the value of a class variable:

```
ListInterface class
    initialize
        "Create a menu. Create constants."
        ListMenu := Menu labels: #('add' 'remove')
    menu
        "Return the list menu."
        ^ListMenu
ListInterface
    hasMenu
        "Return true if a menu is defined."
        ^self class menu notNil
    performMenuActivity
        "Perform the mouse-based activity for my view."
        self hasMenu
            ifTrue: [^self class menu startUp].
```

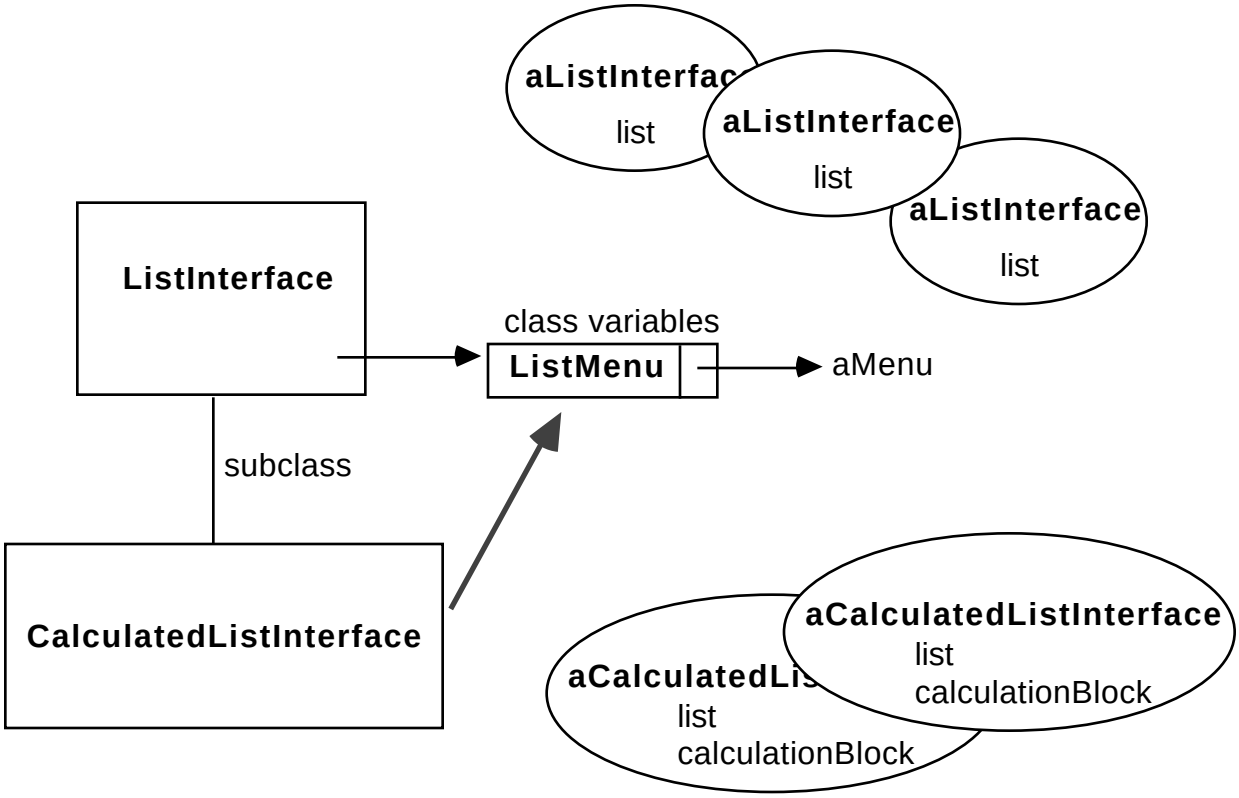


Figure 2. Coding conventions increase the reusability of classes implemented with class variables.

This coding convention reduces the number of direct references to a class variable, as illustrated in Figure 2. It is easier to create subclasses because only the methods that set and retrieve the class variable need to be overridden. Now the code for CalculatedListInterface looks like this:

```
CalculatedListInterface class
    initialize
        "Create a a computed list menu."
        CalculatedMenu :=  Menu labels: #('add' 'remove'
'print')
    menu
        "Return the list menu."
        ^CalculatedListMenu
```

This coding convention effectively restricts the references to a class variable. Because of the nature of the data stored in class variables, it is best for class methods to store and retrieve the class variables. In effect, we have eliminated the sharing between classes and instances.

By eliminating this sharing we have made ListInterface more reusable, but ListInterface still has another problem. Another class variable had to be created by the subclass in order to provide a different menu. Now CalculatedListInterface has two class variables, one of which (ListMenu) is not used.

The root of the remaining problem is that class variables are shared by a class and its subclasses. In our example (and in many other situations) this sharing is inappropriate. Instead, a subclass needs to be able to override inherited data. Class variables share the data between subclass and superclass, so it's not possible for a subclass to override the data. Let's explore another mechanism, class instance variables, that will solve our problem.

**What are class instance variables?** Class instance variables are instance variables that belong to a class. Smalltalk systems rely on this facility. For example, each class stores its name in a class instance variable. Just as each instance has its own values for instance variables, each class has its own values for class instance variables. Unlike class variables, these variables are not shared by all the instances of a class.

Only class methods can reference class instance variables. Direct references to these variables is not allowed from instance methods. Instances methods that need the information stored in a class instance variable must send a message to a class method, which can return the requested information.

Class instance variables, but not their values, are inherited. Because each class has its own values for class instance variables, there is no sharing between a class and subclasses.

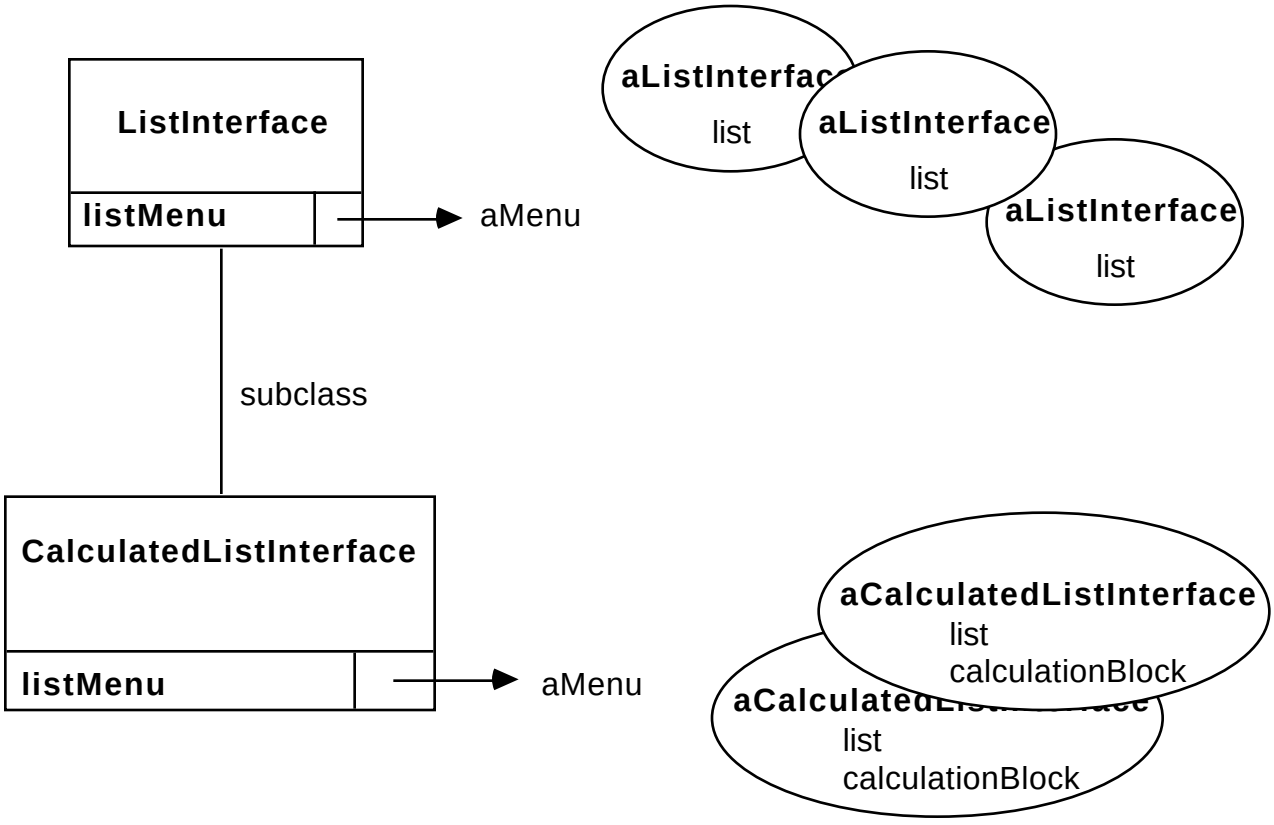


Figure 3. Instances of a class do not directly reference class instance variables and subclasses have their own copy of class instance variables.

This version of ListInterface, illustrated in Figure 3, defines its menu with a class instance variable. The class methods in ListInterface directly reference the class instance variable. Instance methods cannot directly reference listMenu, but instead send messages to the class to access the value of listMenu:

```
ListInterface class
    initialize
        "Create a menu."
        listMenu := Menu labels: #('add' 'remove')
    menu
        "Return the menu."
        ^listMenu

ListInterface
    hasMenu
        "Return true if a menu is defined."
        ^self class menu notNil
    performMenuActivity
        "Perform the mouse-based activity for my view."
        self hasMenu
            ifTrue:[^self class menu startUp].
```

Now let's create a version of CalculatedListInterface that has a different menu. What does the developer need to do? The developer does **not** need to define a new variable. Each class has its own copy of the class instance variable listMenu. Class methods in CalculatedListInterface simply need to assign the appropriate menu to the class instance variable. How many methods need to be overridden? Only one.

```
CalculatedListInterface class
    initialize
        "Create a menu for calculated lists."
        listMenu := Menu labels: #('add' 'remove' 'print')
```

CalculatedListInterface has its own copy of the menu that is stored in the class instance variable listMenu. All methods that access this class instance variable work properly in subclasses because they reference the menu that is stored in their own class. This version of CalculatedListInterface contains only one method, and uses all its defined variables, unlike the previous version that contained a class variable from the superclass.

Most classes, especially those created as standalone abstractions, should use class instance variables so that new subclasses can be created with minimal effort. A common mistake is to use class variables in places where sharing between a class and its subclasses is inappropriate.

**Which version of ListInterface is more reusable?** The version of the class ListInterface that implements the menu with a class instance variable is more reusable than the version that uses a class variable. Fewer methods need to be overridden in order to create a subclass with a different menu. The version implemented with class variables requires a new class variable, while the version implemented with class instance variable does not.

Class instance variables are an important part of Smalltalk because they provide an important mechanism by which more reusable classes are created. All Smalltalk dialects have class variables, but only Smalltalk-80 derived dialects contain class instance variable support as delivered by the vendor. However, users can Smalltalk/V can be extended to support user defined class instance variables with just a handful of methods.

Serious developers of reusable Smalltalk code should use the coding conventions discussed in this article, and class instance variables whenever possible. Class variables should be used only when there is an explicit need for shared variables, because they limit the reusability of classes.