Class Instance Variables for Smalltalk/V

Juanita J. Ewing

Instantiations, Inc.

Derived from Smalltalk Report

In my last column, I described the effect of class variables and class instances variables on class reusability and concluded that classes implemented with class instance variables are more reuseable classes implemented with class variables. Smalltalk-80 derived versions of Smalltalk have class instance variables, but Smalltalk/V versions do not. This column contains the code to add class instance variables to Smalltalk/V Windows.

All objects in Smalltalk/V have instance variables, even class objects. The code in this column just makes the facility apparent for classes, and allows users to define new class instance variables.

How to Define Class Instance Variables

Ordinarily, users see a class definition in a browser like the example in Figure 1. After the code from this column is added to an image, users will see an extended class definition in the browser. The extended class definition consists of two messages, one to the class and one to the metaclass. Figure 2 is an example of an extended class definition with no class instance variables. The message argument to the metaclass is an empty string.

```
Object subclass: #AnimatedObject
instanceVariableNames:
    'position oldPosition jumpIncrement direction ... goCount '
classVariableNames: ''
poolDictionaries:
'WinConstants '
```

Figure 1. Class Definition for AnimatedObject

```
Object subclass: #AnimatedObject
instanceVariableNames:
    'position oldPosition jumpIncrement direction ... goCount '
classVariableNames: ''
```

```
    poolDictionaries:
        'WinConstants '.
    AnimatedObject class instanceVariableNames: ''
```

Figure 2. Extended Class Definition for AnimatedObject

Adding a class instance variables is just like adding an instance variable. The user modifies the argument to the message instanceVariableNames:. The argument is a string containing names of class instance variables. Then the user uses the menu to save the class definition. The system redefines the class and recompiles as needed. Figure 3 is an extended class definition with a class instance variable named defaultDirection.

```
    Object subclass: #AnimatedObject
    instanceVariableNames:
        'position oldPosition jumpIncrement direction ... goCount '
    classVariableNames: ''
    poolDictionaries:
        'WinConstants '.
    AnimatedObject class instanceVariableNames: 'defaultDirection'
```

Figure 3. Extended Class Definition for AnimatedObject with a Class Instance Variables.

The Implementation of Class Instance Variables

The code to add class instance variables to Smalltalk/V Windows consists of five methods, four of which are fundamental and one that is a modification to the class hierarchy browser. After a method by method discussion, a complete listing of the code is included at the end of this column.

Other versions of Smalltalk/V have different implementations, and a different version of the code is necessary to implement class instance variables.

MetaClass class subclassOf: aClass

Modified

This is the instance creation method for MetaClass, and is a private method. It is modified so new instances of meta class have the structure of their superclass. In the original version of this method, each meta class was created with the structure of the Class class.

MetaClass methods instanceVariableNames: stringOfInstVarNames

New

This is a new method representing the public interface for class instance variables. This method is used to redefine the instance variables for a class (class instance variables). The argument to this method is a string containing names of class instance variables. The argument is the same format as for instance variables and class variables.

Class fileOutOn: aStream

Modified

This method has been modified to also write the definition for class instances variables. The result of this method is also used to print the definition of a class in the browser. The string defining class instance variables always prints even if there are no class instance variables. This is necessary because the evaluation of a class definition in the browser must return the same result.

Class recreate: numberOfExtraFields

New

This new private method is used to recreate the class object when the number of class instance variables has changed. It deals with a number of implementation details, such as storing the new class in the Smalltalk dictionary and the global variable TableOfClasses, and the insertion of the new class into the class inheritance hierarchy.

ClassHierarchyBrowser acceptClass: aString from: aPane

Modified

This method has been modified to update the reference to the selected class after saving a new definition of a class. If the number of class instance variables has changed, then a new class object will be created and the browser needs to be updated. This is a private method.

The complete listing of methods for Smalltalk/V Windows follows:

MetaClass class

subclassOf: aClass

```
"Private - Answer a new metaclass that is a subclass of the metaclass for
aClass."
| newMeta |
newMeta := self new.
newMeta
        assignClassHash;
        structure: aClass class structure;
        superclass:
                (aClass == Class
                        ifTrue: [Class]
                        ifFalse: [aClass class]);
methodDictionaries:
        (Array with: (MethodDictionary newSize: 2)) ,
        newMeta superclass methodDictionaries.
^newMeta
```

MetaClass

instanceVariableNames: stringOfInstVarNames

"Define (or redefine) the set of class instance variables for the class which is an instance of this metaClass. The number of class instance variable may be increased only if there are no existing instances of the class."

```
| theClass oldSize newSize aStream theClassName |
theClass := self instanceClass.
theClassName := theClass symbol.
oldSize := self instVarNames size.
newSize := stringOfInstVarNames asArrayOfSubstrings size.
oldSize < newSize
    ifTrue:
        [" if the size of the class object needs to increase
        there must be no instances"
        theClass withAllSubclasses do:
            [:aClass | aClass allInstances notEmpty
                ifTrue: [^self error: 'Has instances']]].
self instVarNames: stringOfInstVarNames.
oldSize < newSize
    ifTrue:
        [theClass recreate: newSize-oldSize
        "recreate the class object"].
theClass := Smalltalk at: theClassName.
aStream := WriteStream on: (String new: 64).
theClass fileOutOn: aStream.
Smalltalk logSource: aStream contents forClass: theClass.
self compileAll.
self allSubclasses do:
    [:aClass | aClass compileAll].
^theClass
```

Class

fileOutOn: aStream

"Append the extended class definition message for the receiver to aStream. Include the statement for the definition of class instance variables."

```
| aString |
aStream cr;
nextPutAll: self superclass printString; space;
nextPutAll: self kindOfSubclass; space;
nextPutAll: name storeString; cr; space; space.
self isBits
    ifFalse:
        [aStream nextPutAll: 'instanceVariableNames: '
        (aString := self instanceVariableString) isEmpty
            ifFalse: [aStream cr; nextPutAll: ' '].
```

```
            aStream
                nextPutAll: aString storeString;
                cr; space; space].
    aStream nextPutAll: 'classVariableNames: '.
    (aString := self classVariableString) isEmpty
        ifFalse: [aStream cr; nextPutAll: ' '].
    aStream
        nextPutAll: aString storeString;
        cr; space; space;
        nextPutAll: 'poolDictionaries: '.
    (aString := self sharedVariableString) isEmpty
        ifFalse:[ aStream cr; nextPutAll: ' '].
    aStream nextPutAll: aString storeString.
    "Include class instance variable definition."
    aString := self class instanceVariableString.
    aStream nextPut: $.; cr.
    aStream nextPutAll: self class name.
    aStream nextPutAll: ' instanceVariableNames: '.
    aStream nextPutAll: aString storeString.
    aStream cr; space; space
```

Class

recreate: numberOfExtraFields

"Private - Replace this class object with an identical object with additional fields for class instance variables."

```
    | newInstance mySuperclass myName oldId |
    myName := self symbol.
    newInstance := self class basicNew.
    oldId := self id.
    1 to: self class instSize - numberOfExtraFields
        do:
            [:i|
            newInstance instVarAt: i put: (self instVarAt: i)].
    mySuperclass := self superclass.
    mySuperclass removeSubclass: self.
    mySuperclass addSubclass: newInstance.
    Smalltalk at: myName put: newInstance.
    newInstance methodDictionary do:
        [ :m |
        m classField = self
            ifTrue: [m classField: newInstance]].
    newInstance subclasses copy do:
        [:sub |
        sub superclass: newInstance.
        sub recreate: numberOfExtraFields].
    TableOfClasses at: oldId + 1 put: newInstance.
```

```
newInstance id: oldId.
self become: DeletedClass
```

ClassHierarchyBrowser

acceptClass: aString from: aPane

"Private - Accept aString as an updated

class specification and compile it. Notify aPane if the compiler detects errors."

```
| result isClass |
result := Compiler
    evaluate: aString
    in: nil class
    to: nil
    notifying: aPane
    ifFail: [^true].
Smalltalk logEvaluate: aString.
isClass := result isKindOf: Class.
isClass
    ifTrue: [selectedClass := result].
self changed: #instanceVars:.
^isClass not
```