How Should Classes Be Initialized?

Juanita J. Ewing
Instantiations, Inc.

Derived from Smalltalk Report

Class initialization should be systematic and predictable. Have you ever sent messages to a class and gotten strange errors related to uninitialized class data? Uninitialized class data is a result of Smalltalk programming conventions and lack of support by the programming environment. Some minor changes in the Smalltalk programming environment could greatly improve this situation.

What needs to be initialized? Smalltalk classes have two or three different kinds of class data that must be initialized:

- class variables,
- class instances variables, and
- pool dictionaries.

Smalltalk-80 derived dialects have class instance variables, but Smalltalk/V dialects do not. Each kind of class data has semantic differences.

Classes can have class variables, which are shared between all instances and the class. Class variables can be referenced from both instance and class methods simply by referring to the name of the class variable.

Class instance variables are storage for the class and can be referenced only from class 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.

Pool dictionaries are shared between several classes. The keys in a pool dictionary can be directly referenced from both instance and class methods.

Which of these is inherited? All Smalltalk programmers are familiar with the inheritance semantics of instance variables. The variable is inherited, but not its value.

Unlike instance variables, a class variable and its value is inherited. That means that the value of a class variable is shared with subclasses and all of their instances. Figure 1 diagrams a class variable defined by class `A`. The subclass `B` inherits the class variable. Instances methods from the subclass and superclass are able to reference class variables.

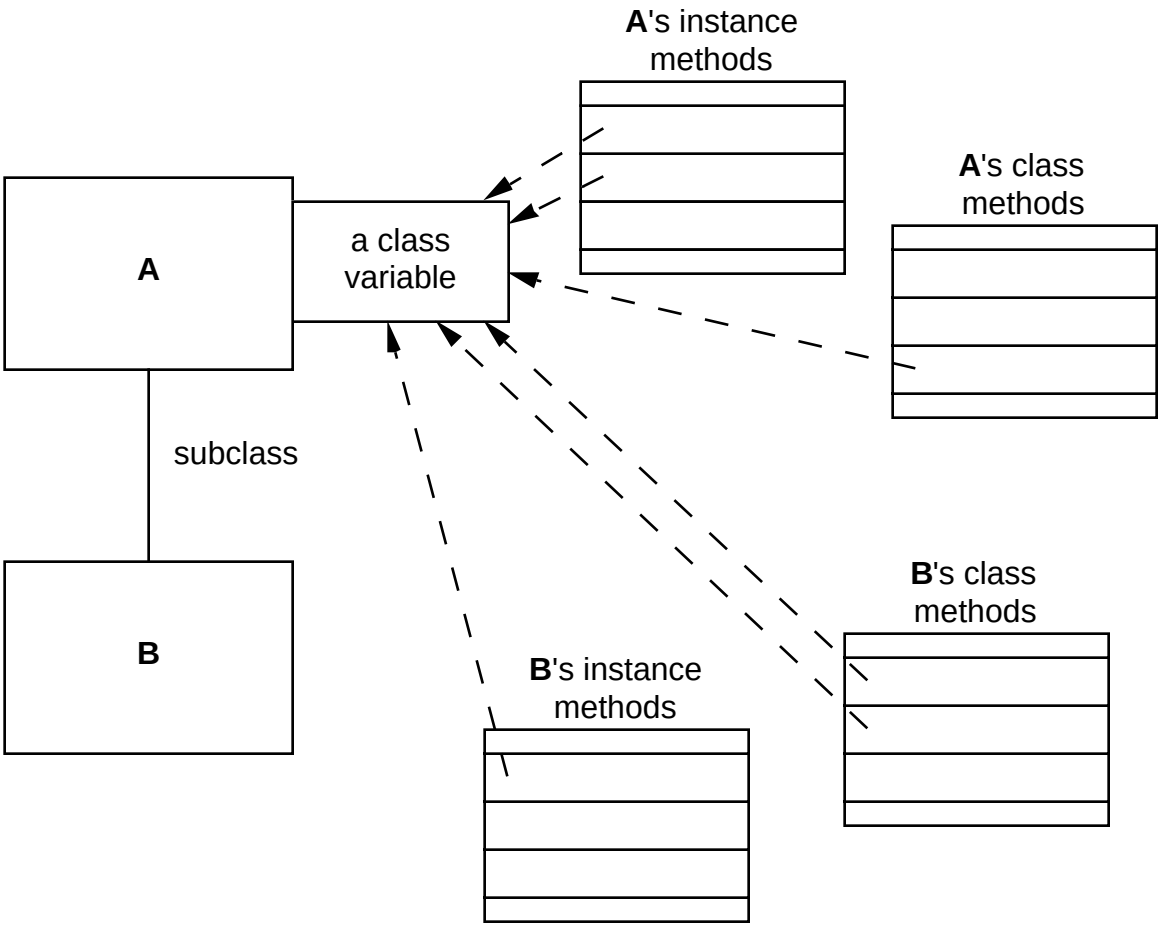

Figure 1. Class Variables

Class instance variables are much like the instance variables we use all the time in Smalltalk programming, with one exception: class instance variables are instance variables for a class instead of for instances of a class. The semantics of class instance variable are similar to those of instance variables. The variable is inherited, but not its value. Each class must fill in its own value. These kind of variables are handy because subclasses can easily override values defined in a superclass.

In Figure 2, the class A defines a class instance variable, i. The subclass B inherits the class instance variable i, and defines another class instance variable. Only class methods can refer to class instance variables. The receiver of the message determines which storage slot will be referenced. For example, suppose the method initialize references the class instance variable i. If the receiver of the message initialize is A, then the storage slot in the class A will be referenced. If the receiver is B, then the storage slot in class B will be referenced.

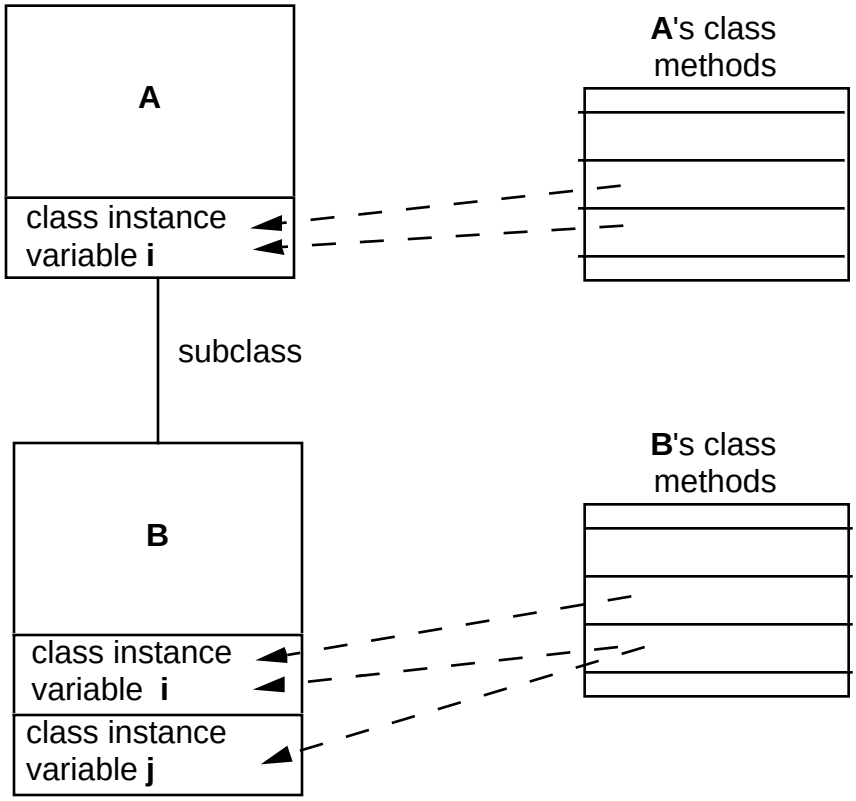

Figure 2. Class Instance Variables

Pool dictionaries are similar to class variables. Pool dictionaries and their values are inherited by subclasses. In Figure 3, the class A defines a pool dictionary. Both class and instance methods can reference keys in the pool dictionary. The subclass B inherits the pool dictionary, including its values. Instance and class methods from the subclass B can reference keys in the pool dictionary.

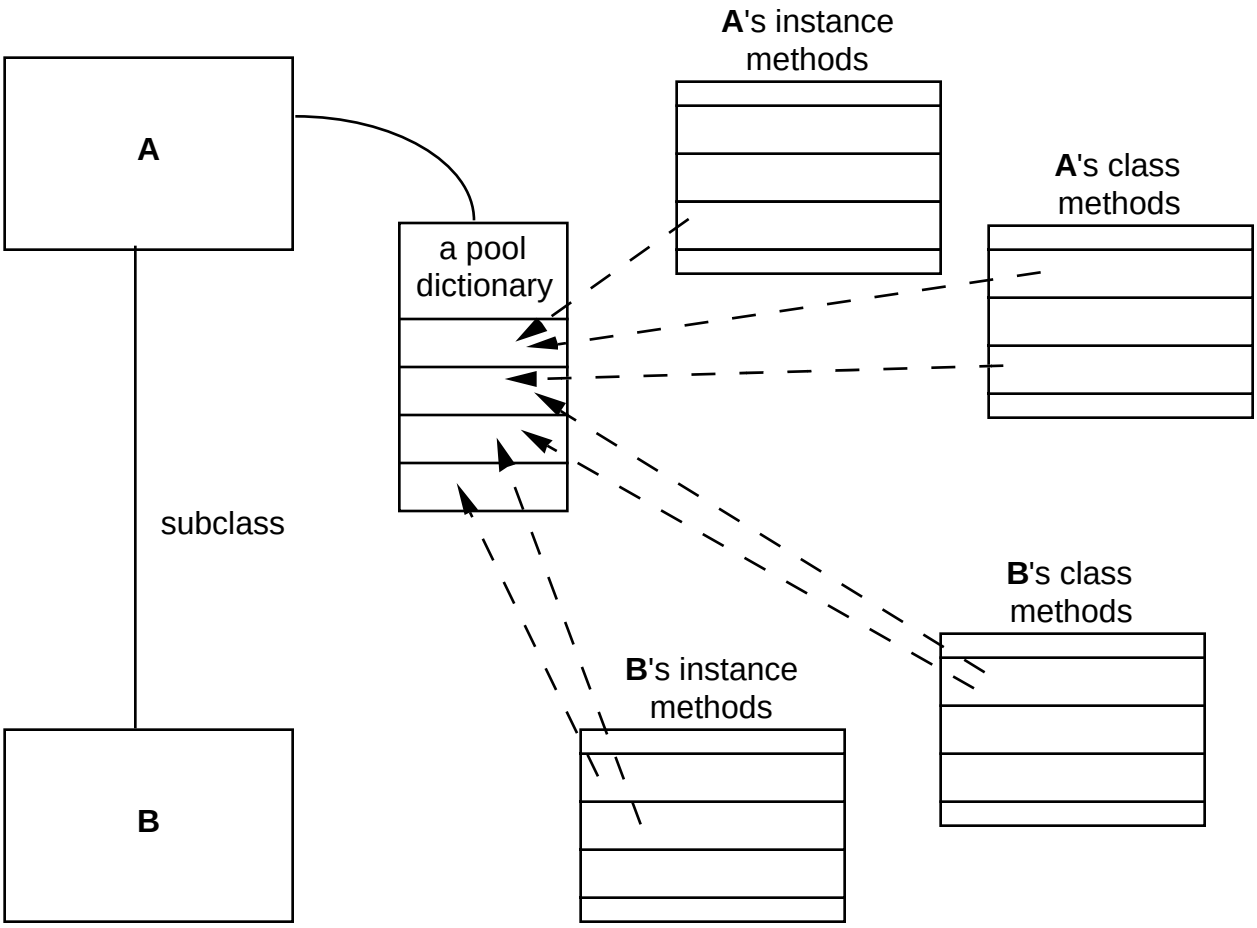

Figure 3. Pool Dictionaries

Let's examine the inheritance consequences.

What happens if class methods are used to initialize? Class data needs to be initialized. The most common practice is to use a class method called initialize. This method typically is used to initialize all class data, no matter what kind of class data it is.

Assume an initialization method initializes a class variable. The class that defines the class variable needs to execute the initialization method. Since the value of a class variable is inherited, subclasses don't need to execute this method. In fact, it may be an error to do so because some valuable data may have accumulated in a class variable. Subclasses inherit the initialize method, but should not execute this method. This situation is a violation of good object-oriented programming.

Assume the initialize method initializes a class instance variable. The value of this variable is not inherited, so subclasses must execute the initialize method to initialize the variable. Either an inherited method or a local initialization method can be used. The initialization method must be executed by each class.

Pool dictionaries are shared between several classes. The current Smalltalk convention is for one of the classes to provide an initialization method. Should the initialization method be executed by subclasses? No. It may wipe out valuable accumulated data. This case is analogous to the situation with class variables.

Does Smalltalk have initialization conventions? Smalltalk has a convention for the initialization of instances which is to invoke the superclasses' initialization method if subclasses must override it. The convention arose because the superclass can initialize variables. Subclasses avoid duplicating the inherited code.

Initialization methods typically look like this:

```
initialize
    "Invoke the receiver's inherited method.
    Initialize my variable to the integer 2."

super initialize.
myVariable := 2
```

Can we apply this convention to class initialization? The super initialize convention doesn't work well with class initialization methods. A Smalltalk programmer can't tell if the initialization method should be executed without examining the code. If only class instance variables are initialized, this convention works. If class variables or pool dictionaries are initialized, then this convention doesn't work since the values of these variables are inherited.

Smalltalk programmers don't restrict their initialization methods. They use initialize methods for all kinds of class data. Therefore, the existence of an initialize method in a hierarchy is not a good indication of initialization requirements.

Do class initialization methods work? Even if you ignore the inheritance issues and the super initialize convention problems, there are still flaws in class initializations contained in class methods. Execution of the initialize method is an action that is separate from the compilation of the initialize method. This separation leads to another problem. How many times have you edited an initialize method but forgotten to execute it?

If you file in someone else's class, it may or may not have a do–it to perform an initialize. Dialects of Smalltalk-80 try to get around this problem by automatically filing out an initialize do–it when a class containing an initialize method is filed out. When the class is filed back in, the do–it is executed. This heuristic fails in two common cases. When a developer creates a method for initialization and calls it something other than initialize, such as initializeVariables, then the programming environment fails to detect the purpose of the method and does not treat it specially. Because this heuristic only examines behavior in a single class, it also fails when an inherited initialization method needs to be executed by a subclass.

Should inherited methods execute without error? Any inherited method should be able to execute without error. Dialects of Smalltalk-80 override inappropriate inherited methods with an implementation consisting of self shouldNotImplement. High quality class hierarchies should never allow users to execute methods that create errors. High quality programming environments should not encourage the construction of code that creates these errors. Unfortunately, Smalltalk class initialization conventions promote the inheritance of inappropriate methods.

The Smalltalk programming environment has a quality problem.

Should class initialization be inherited? No. Since it is impossible to tell if a class initialization method should be inherited without examining the code, class initialization methods should never be inherited. Initialization methods too frequently contain references to inherited class variables and pool dictionaries. It is too easy to make a mistake and execute an inherited initialization method that is inappropriate.

The code that performs class initialization should be compiled in the scope of the class in order to reference class data, but it doesn't have to be a class initialization method. The Smalltalk programming language has enough power and flexibility to provide another mechanism in the programming environment for class initialization.

How should classes be initialized? Class initialization should not be a class method. Instead, classes should have a separate component that contains the class initialization code. We will call this the *class initialization*.

The class initialization should be bundled with the class and supported by the programming environment as a part of the class. The code that performs the initialization should be able to reference all class data: class variables, class instance variables and pool dictionaries. The initialization code should be executed so that self is bound to the class. These characteristics are

also characteristics of class methods, so programmers don't have to change the way they write initialization code. They just have to designate it as class initialization code.

Separate class initialization code is beneficial in several ways. Inheritance of inappropriate class methods avoids costly errors. Functionality can be added to the programming environment that improves productivity. When the code to initialize a class is identified, the programming environment can take special action to support its intended functionality. For example, when the class initialization is redefined, it could be automatically executed by the environment. The class initialization could automatically be filed out when a class is filed out, and executed when the class is filed back in. If just these actions are supported by the programming environment, then much time would be saved by Smalltalk programmers.

All of this can easily be implemented in a Smalltalk programming environment. The result is separate class initialization whose purpose is known by the programming environment. This kind of class initialization is not inherited and so avoids errors. A class initialization can send messages to the class, and in doing so may execute class methods.