Tyler Hackett

Professor James Finley

CIS-18B

16 March 2017

Singletons: Inherent Problems & Ways to Revise Them

The topic of singletons is perhaps one of the most contentious topics of the modern programming era, and it's easy to see why; the singleton is a fantastic way to force you into a tightly coupled environment, the idea of a “single instance” can be confusing in certain contexts, it complicates behavior during unit testing, and it spits in the face of the object oriented paradigm. However, despite the many pitfalls of the pattern, its use is inevitable – not in the the sense that there aren't better alternatives, but in the sense that programmers will continue to implement singletons regardless of how ill advised it may be. For that reason, this paper will focus on potential ways to improve the concept of the singleton.

Let's start off with a simple point: Singletons are not a design pattern. While most design patterns are geared towards giving the programmer a powerful and abstract idea to expand on, the vast majority of singletons are written exactly the same and don't capitalize on these common design traits. This is not necessarily a bad thing, but it does mean singletons should not be treated as a design pattern if we want to use them productively. If there is any hope in freeing up some of the burden of singletons, they must be implemented more rigorously than the typical design pattern. Naturally, the first question to ask yourself is how a more constricted singleton can be achieved, and one might come to the conclusion that singletons should be a native component of languages that intend to support them.

To get a feel for how this might work, look no further than Python meta-classes. By abstracting the singleton behavior into a lower level than the class itself, we can standardize the

behavior for singletons across any desired environment, gain the ability to intuitively subclass singletons, and create more elegant singletons with appreciably less duplicate code (*Robinson, garyrobinson*). Taking this a step further, if a language defines a singleton as a special type of class (similarly to interfaces in Java, for example), the language can enforce any desirable restrictions to singletons. Such restrictions might include requiring all fields to be immutable in order to avoid global states. Another good example of this can be found in the Scala language; Scala provides the object type to define a native singleton.

Admittedly, Scala objects behave more like static classes than true singletons (*EPFL*), but this reinforces the idea of allowing each language to define the singleton in a way that will be unambiguous for programmers. Consider the case of languages like PHP -- since PHP runs many concurrent processes, the idea of a "single instance" becomes vague and confusing since each process will have its own independent instance of the supposed singleton. For this reason, singletons in PHP are highly discouraged (*Edgell, Classically*). This confusion would vanish if singletons were their own type in PHP.

The singleton will remain a perpetual bane of many programmers; some will never be convinced that it is anything more than an the anti-pattern from hell. But with some thoughtful considerations, singletons could ultimately become more good than bad. Until singletons find their rightful place in the framework of programming languages, we will never be able to see the full potential they have to offer.

Works Cited

Edgell, Jeremy. “Singletons in PHP”, *Classically,* 18 March 2015*,*

http://classically.me/content/singletons-php

École Polytechnique Fédérale de Lausanne. “Singleton Objects”, *EPFL*, 2015,

http://docs.scala-lang.org/tutorials/tour/singleton-objects.html

Robinson, Gary. “Python Singleton Mixin Class”, 11 March 2004,

http://www.garyrobinson.net/2004/03/python_singleto.html