Professional Expression Assignment 1

CISC 640 – Operating Systems

Professor Greg Simco

Nova Southeastern University

Eric Webb

When it comes to the two computer science professionals who are having a heated discussion on implementation details, discussing whether they are to be exposed or abstracted away, my personal opinion belongs with that of computer science professional A. The one in favor of the implementation details being abstracted away and not exposed.

I have come to this conclusion because it is a well-known fact that abstraction is one of the four pillars of object oriented programming (Vivek, n.d). Abstraction allows for you to hide the implementation details and only expose the result functionality. This is of course beneficial because it adds another level of security to your architecture by hiding the implementation details. In layman terms, if you have a secret recipe for some baked goods, you would not expose the details of your special ingredients and processes, but keep them protected. All your customers would know is that your final product was delicious rather than having details of the ingredients and cooking methods. In this particular case, the secret ingredients would be the implementation details that have been abstracted away and are no longer visible.

I do not agree with the case of computer scientist B, the professional who wanted to expose the implementation details. Exposing this would be an inherit security flaw. If you have proprietary data, you do not want to expose that to the outside world. That is why it is better to abstract those details away so they cannot be viewed.

So how does one abstract away implementation details? Probably one the best and most infamous ways to explain this is through the use of Java abstract classes and interfaces (Miglani, 2019). Java interfaces allow you to fully utilize abstraction. Abstract classes provide an option for either partial of full abstraction. Interfaces provide full abstraction by only allowing abstract methods. Meaning, they do not have a method body only a method signature. This ensures that when using interfaces, 100% of your method implementation details have been abstracted away. Abstract classes can be partially abstract because they allow for both abstract methods and non-abstract methods. This effectively abstracts away method bodies on some methods while exposing them on others (Miglani, 2019). Typically, I like to explain that interfaces are for implementing verbs, while abstract classes are for inheriting nouns. For example, you can have an abstract class called animal, with some classes that inherit this animal abstract class. This could be portrayed as a dog class and a snake class. Both have animal methods, whose implementations have been abstracted into their specific instances of dog and snake. Building from this, not only can dog and snake inherit the animal class abstract methods, but at the same time they both can implement an interface. For this scenario we will call our interface eats. The eats interface abstracts the details of the eats method to be designated by one of the classes implementing it. Since both dog and snake eat differently, each ones eats method that it implements will have the same method signature, but a different method body. We do not care how or what the snake and dog eat, as long as they do in fact eat and they are in fact animals. That is the beauty of abstraction, hiding the details but exposing the result.

In conclusion, I chose computer scientist A to be the correct side of this debate because it is a better architectural solution to hide your implementation details on all things, then to do the opposite and expose them to the world. This is a core concept of object oriented programming and can be applied through the use of Java abstract classes and interfaces along with other OOP based languages.

Work Cited.

Miglani, G. (2019, February 21). Abstraction in Java. Retrieved August 20, 2019, from

<https://www.geeksforgeeks.org/abstraction-in-java-2/>

Vivek, H. (n.d.). 4 Pillars of OOP. Retrieved August 20, 2019, from

<http://www.corejavaguru.com/java/oop/4pillars>