This is a graded discussion: **10 points possible** due Apr 8

Discussion 1: Class Inheritance vs Object Composition

2

In this week's discussion, we will be discussing class inheritance and object composition. Class inheritance and object composition are the two most common techniques for reusing functionality.

Inheritance defines classes in terms of existing classes; a child class inherits data and operations from its parent class. The child class can then override parent-class operations, if permitted, and add some of its own.

Composition, however, is implemented as an object containing internal references to other objects and interacting with other objects through interface members. Composition thus depends on relationships between objects not inheritance.

The Second Principle of OO Design states "Favor object composition over class inheritance."

Do you agree or disagree? Why? Please give us your opinion on this issue and critique at least one other student's opinion.

Search entries or author | Unread | Subscribed

Reply

Eyob Bayou
6:08pm

Yes I do agree with the "Favor object composition over class inheritance" principle. The next three points are what I understand to be the advantages of composition:
1. It helps to easily model a system. Composition is more natural because many real world objects are easily described by their interactions with other objects than their hierarchical relationships (inheritance). And in my experience, I find many base classes to be some form of abstraction of real world objects. Those abstractions could be naturally represented by interfaces as in composition than base classes as in in inheritance.
2. Composition allows greater flexibility by substantially reducing implementation dependencies, i.e. by its support of polymorphism. Objects and behaviors could be dynamically picked during run time. This allows for flexibility as the binding is dynamic as opposed to static. In other words, the use of interfaces means following the other OO design principle called "program to an interface, not an implementation" which is the principle that ensures flexibility by allowing objects of the same type (interface) to be used interchangeably.
3. The other advantage that I've confirmed in my coding experience is that composition makes programs highly extensible. If there are other business level behaviors that need to be added later, composition is a good choice because new interfaces could be easily defined for the new behaviors without going for the surgical process of capturing these new behaviors with in the hierarchical structure of inheritance.
4. As it's the case with any approach, compositions isn't without disadvantages. The typical one is due to the fact that it uses interfaces in place of base classes. Interfaces don't have implementations - making them inconvenient when it comes to identical behaviors shared across multiple derived classes. Each derived class is supposed to implement its own behavior which leads to same behavior getting implemented repetitively. This significantly affects code reuse.

Reply

Brian Sengsourichanh
9:33pm

I agree with Eyob on all points, in that, the benefits granted by way of object composition (a more flexible and extensible system that is easier to architect) outweigh its negatives (the possibility for duplicate implementations across multiple subtypes) and as a result is favored over class inheritance.

Piggybacking off of Eyob's great summation of the advantages of composition, I'd also like to highlight that composition allows for a more maintainable program; one more resistant to future changes than another going the route of class inheritance. Though the ability to inherit base implementations by way of inheritance is a powerful trait, the possible need to begin overriding these inherited methods may arise. Over time, overriding inherited methods willy nilly can eventually lead to a situation where other inherited methods may be expecting the overridden method to behave in a specific way and thus cause unintended side effects to crop up.

Reply