Inheritance and Polymorphism - Ease of use vs Purity

Asked 16 years, 4 months ago Modified 14 years, 11 months ago

Viewed 2k times

6

In a project our team is using object lists to perform mass operations on sets of data that should all be processed in a similar way. In particular, different objects would ideally act the same, which would be very easily achieved with polymorphism. The problem I have with it is that inheritance implies the *is a* relationship, rather than the *has a* relationship. For example, several objects *have a* damage counter, but to make this easy to use in an object list, polymorphism could be used - except that would imply an *is a* relationship which wouldn't be true. (A person *is not a* damage counter.)

The only solution I can think of is to have a member of the class return the proper object type when implicitly casted instead of relying on inheritance. Would it be better to forgo the *is a* / *has a* ideal in exchange for ease of programming?

Edit: To be more specific, I am using C++, so using polymorphism would allow the different objects to "act the same" in the sense that the derived classes could reside within a single list and be operated upon by a virtual function of the base class. The use of an interface (or

imitating them via inheritance) seems like a solution I would be willing to use.

c++ inheritance oop polymorphism

Share

Improve this question

Follow

edited Sep 21, 2008 at 7:38

Jason Etheridge

6,897 ● 5 ● 31 ● 33

asked Aug 15, 2008 at 1:00

Cristián Romo

10k ● 12 ● 52 ● 50

10 Answers

Sorted by: Highest score (default)

5

I think you should be implementing interfaces to be able to enforce your *has a* relationships (am doing this in C#):

```
public interface IDamageable
{
    void AddDamage(int i);
    int DamageCount {get;}
}
```

You could implement this in your objects:

```
public class Person : IDamageable

public class House : IDamageable
```

And you'd be sure that the DamageCount property and has a method to allow you to add damage, without implying that a person and a house are related to each other in some sort of heirarchy.

Share Improve this answer

Follow

answered Aug 15, 2008 at 1:54

Jon Limjap

95.3k ● 15 ● 103 ● 153

3

This can be accomplished using multiple inheritance. In your specific case (C++), you can use pure virtual classes as interfaces. This allows you to have multiple inheritance without creating scope/ambiguity problems. Example:

```
class Damage {
    virtual void addDamage(int d) = 0;
    virtual int getDamage() = 0;
};

class Person : public virtual Damage {
    void addDamage(int d) {
        // ...
        damage += d * 2;
    }

    int getDamage() {
        return damage;
    }
};

class Car : public virtual Damage {
    void addDamage(int d) {
        // ...
        damage += d;
    }
```

```
    int getDamage() {
        return damage;
    }
};
```

Now both Person and Car 'is-a' Damage, meaning, they implement the Damage interface. The use of pure virtual classes (so that they are like interfaces) is key and should be used frequently. It insulates future changes from altering the entire system. Read up on the Open-Closed Principle for more information.

Share Improve this answer

Follow

answered Aug 15, 2008 at 3:41

Brian

771 ● 1 ● 6 ● 18

1

I agree with Jon, but assuming you still have need for a separate damage counter class, you can do:

```
class IDamageable {
  virtual DamageCounter* damage_counter() = 0;
};
class DamageCounter {
  ...
};
```

Each damageable class then needs to provide their own damage_counter() member function. The downside of this is that it creates a vtable for each damageable class. You can instead use:

```
class Damageable {
 public:
  DamageCounter damage_counter() { return damage_count
 private:
  DamageCounter damage_counter_;
};
```

But many people are **Not Cool** with multiple inheritance when multiple parents have member variables.

Share Improve this answer

Follow

answered Sep 16, 2008 at 9:56

0124816

1,083 ● 6 ● 11

0

Sometimes it's worth giving up the ideal for the realistic. If it's going to cause a massive problem to "do it right" with no real benefit, then I would do it wrong. With that said, I often think it's worth taking the time to do it right, because unnecessary multiple inheritance increases complexity, and it *can* contribute to the system being less maintainable. You really have to decide what's best for your circumstance.

One option would be to have these objects implement a `Damageable` interface, rather than inheriting from `DamageCounter`. This way, a person *has-a* damage counter, but *is* damageable. (I often find interfaces make a lot more sense as adjective than nouns.) Then you could have a consistent damage interface on `Damageable` objects, and not expose that a damage counter is the underlying implementation (unless you need to).

If you want to go the template route (assuming C++ or similar), you could do this with mixins, but that can get ugly really quickly if done poorly.

Share Improve this answer

Follow

answered Aug 15, 2008 at 1:13

Derek Park

46.8k ● 16 ● 59 ● 76

0

This question is really confusing :/

Your question in bold is very open-ended and has an answer of "it depends", but your example doesn't really give much information about the context from which you are asking. These lines confuse me;

> sets of data that should all be processed in a similar way

What way? Are the sets processed by a function? Another class? Via a virtual function on the data?

> In particular, different objects would ideally act the same, which would be very easily achieved with polymorphism

The ideal of "acting the same" and polymorphism are absolutely unrelated. How does polymorphism make it

easy to achieve?

Share Improve this answer

Follow

answered Aug 15, 2008 at 1:17

Andrew Grant

58.8k • 22 • 131 • 144

0

@Kevin

> Normally when we talk about 'is a' vs 'has a' we're talking about Inheritance vs Composition.
>
> Um...damage counter would just be attribute of one of your derived classes and wouldn't really be discussed in terms of 'A person is a damage counter' with respect to your question.

Having the damage counter as an attribute doesn't allow him to diverse objects with damage counters into a collection. For example, a person and a car might both have damage counters, but you can't have a `vector<Person|Car>` or a `vector<with::getDamage()>` or anything similar in most languages. If you have a common Object base class, then you can shove them in that way, but then you can't access the `getDamage()` method generically.

That was the essence of his question, as I read it. "Should I violate `is-a` and `has-a` for the sake of treating certain objects as if they are the same, even though they aren't?"

Share Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 • 1

answered Aug 15, 2008 at 1:22

Derek Park

46.8k • 16 • 59 • 76

0

Normally when we talk about 'is a' vs 'has a' we're talking about Inheritance vs Composition.

Um...damage counter would just be attribute of one of your derived classes and wouldn't really be discussed in terms of 'A person is a damage counter' with respect to your question.

See this:

http://www.artima.com/designtechniques/compoinh.html

Which might help you along the way.

@**Derek**: From the wording, I assumed there *was* a base clase, having re-read the question I kinda now see what he's getting at.

Share Improve this answer

Follow

edited May 23, 2017 at 12:13

Community Bot

1 • 1

answered Aug 15, 2008 at 1:15

Kev

0

"Doing it right" will have benefits in the long run, if only because someone maintaining the system later will find it easier to comprehend if it was done right to begin with.

Depending on the language, you may well have the option of multiple inheritance, but normally simple interfaces make the most sense. By "simple" I mean make an interface that isn't trying to be too much. Better to have lots of simple interfaces and a few monolithic ones. Of course, there is always a trade off, and too many interfaces would probably lead to ones being "forgotten" about...

Share Improve this answer

Follow

answered Aug 15, 2008 at 1:29

Andrew

663 ● 5 ● 11

0

@Andrew

> The ideal of "acting the same" and polymorphism are absolutely unrelated. How does polymorphism make it easy to achieve?

They all have, e.g., one function in common. Let's call it `addDamage()`. If you want to do something like this:

```
foreach (obj in mylist)
    obj.addDamage(1)
```

Then you need either a dynamic language, or you need them to extend from a common parent class (or interface). e.g.:

```
class Person : DamageCounter {}
class Car : DamageCounter {}

foreach (DamageCounter d in mylist)
    d.addDamage(1)
```

Then, you can treat `Person` and `Car` the same in certain very useful circumstances.

Share Improve this answer

Follow

answered Aug 15, 2008 at 1:29

Derek Park

46.8k ● 16 ● 59 ● 76

0

Polymorphism *does not require inheritance*. Polymorphism is what you get when multiple objects implement the same message signature (method).

Share Improve this answer

Follow

answered Jan 18, 2010 at 21:29

Steven A. Lowe

61.1k ● 19 ● 135 ● 204