

Forum:[Hack Forums](#)

Thread: [What is the purpose of Interfaces and Abstract methods???](#)

colchestercastle - October 1, 2016, 12:03 pm

Would it not be more efficient just to define methods that do the same thing in each class and name them slightly differently instead of using an interface?

Also, with abstract methods, you have to redefine them in the class you want to use them in and inherit the class the abstract method is in (If I am using the terminology wrong, please correct me). So why not just define it a single time and every time you want to use the same method just copy and paste and change the name slightly? Thanks for any help

GammaG - October 1, 2016, 12:14 pm

You can let several classes implement an interface in order to make sure every class has the methods you have set in the interface.

As an example you can make sure every class implementing the interface is compatible with your framework.

Also don't be so sure that every class will do the same actions inside the function, otherwise you could just make an static function for that somewhere.

SIM-2228224 - October 1, 2016, 9:51 pm

Its more work doing things in the way you suggested OP. The goal of OOP is scalability, flexibility and code reuse. Not everything works in copy paste fashion, sometimes there are nuances that require a slightly more flexible implementation of an abstract method to accommodate. For example you have a game that uses a weapons class. The weapons class would be abstract, and would have an abstract method named damage. Now there are different types of weapons in a game. Blades, guns and explosives. Each would have to extend the abstract weapon class. But those three objects inflict damage in a specific way. Guns shoot, blades cut, and explosives...well, explode. But they all do so differently. Guns can be either automatic or semiautomatic, blades can be short like a dagger, or long like a sword or projectiles like a shuriken, and explosions can range from small to very large.

That's where interfaces come in, where they can be implemented differently by each type of weapon that either shoots, cuts or explodes in their own way . All these variations can make things pretty hairy, so the use of interfaces and abstractions is a way to manage that complexity. Sometimes it's better to have an umbrella object of which to identify different objects under with common functionalities to use appropriately. The purpose of this is more obvious in large scale systems.

+Haze - October 2, 2016, 10:45 am

(Oct 1, 2016 12:15 PM)lolxd123 Wrote: [\[CITE\]](#) Google "Java Inheritance."

Both of these are a big part of OOP. I don't feel like writing a lecture on a topic that requires a thorough explanation so consider this.

Say for example, you have an interface called shape. This shape interface has a few methods that defines essentially every shape, so you would implement it into your Rectangle, square, etc etc. In some cases such as a circle, you may have to implement methods specific to that shape, however, it is important you implement everything in that interface.

For abstract classes (with abstract methods), let's look at a game example. You are coding a video game let's say you have an abstract monsterEntity (before anyone comes along and says you should have a single entity class, shut up). In this monster entity, you have a constructor and some methods already defined for health, armor, name, etc (anything a monster in a video game would have). Then you have an abstract attack method, every monster has an attack unique to them. Therefore, you must define this method on a monster to monster basis. While you extend this abstract class, you can also use your constructor to utilize other methods in the superclass such as the health, armor, and name methods already defined.

These are two very rough examples with a VERY rough explanation for abstract classes. GOOGLE. Both of which are TERRIBLE examples. Shape wouldn't even be an interface, it should be a class. I don't think you understand Interfaces.

Stealth™ - October 2, 2016, 3:52 pm

(Oct 2, 2016 10:45 AM)+Haze Wrote: [\[CITE\]](#) Both of which are TERRIBLE examples. Shape wouldn't even be an interface, it should be a class. I don't think you understand Interfaces. I don't think you understand them :P

```
Code public interface Shape {  
  
    int numSides();  
  
}  
  
class Square implements Shape {  
  
    @Override  
    public int numSides() {  
        return 4;  
    }  
  
}
```

Certainly it can be both ways, but it's a perfectly valid example.

+Haze - October 2, 2016, 4:53 pm

(Oct 2, 2016 03:52 PM)Stealth™ Wrote: [\[CITE\]](#) I don't think you understand them :P

```
Code public interface Shape {  
  
    int numSides();  
  
}  
  
class Square implements Shape {  
  
    @Override  
    public int numSides() {  
        return 4;  
    }  
  
}
```

Certainly it can be both ways, but it's a perfectly valid example.

I'm pretty sure I do, and yes that's a valid but bad example.

What I mean to say it, using shape as interface isn't really what interfaces were designed for. The whole point of interfaces is to be able to define a common trait between classes that don't necessarily have a direct relationship in the inheritance hierarchy. For example, IAnnoying could be an example of an interface, but different type of class can implment the IAnnoying interface. For example, a car can inherit it and so can a fly or baby. A car can constantly beep its horns, a fly can constantly buzz in your face and a baby can constantly car, all which are annoying traits. But notice how a car, fly and baby have no direct inheritance?

That's why I say Shape should not be an interface, Shape should be a base class, and classes like Rectangle and Triangle etc should all inherit from Shape.

Judging from your example of have a method called numSides(), you're going to have to redefine that in every subclass of the Shape interface. Why do that? You're essentially repeating yourself which is not the aim of object oriented programming(which is proof that you've used interfaces wrong in your example). Now if you had Shape as a class which you should of, you could of defined numSides once and that's that.

I don't know if I've explained myself well enough but I hope someone can understand what I mean.

Note: I did not say you people were incorrect with your explanations, I'm just saying the example you gave was bad, and wasn't really the best example to showcase how an interface should be used.

GenericSkid - October 3, 2016, 11:46 am

(Oct 2, 2016 04:53 PM)+Haze Wrote: [\[CITE\]](#) -snip-
Holy shit, when did HF get in-line pastbins?

Stealth™ - October 3, 2016, 12:21 pm

(Oct 3, 2016 11:46 AM)Lord Pankake Wrote: [\[CITE\]](#) Holy shit, when did HF get in-line pastbins?
Check out the [paste] tag :D

+Haze - October 4, 2016, 2:32 pm

(Oct 2, 2016 09:45 PM)lolxd123 Wrote: [\[CITE\]](#) You explained yourself just fine. However, for teaching purposes only, a shape interface is completely fine in the right context. If you create a shape interface that only defines methods that even in an abstract class, is not default, then you would only need an interface.

For teaching purposes (which you completely disregard), a shape interface that defines area and perimeter would be a valid example. Just because a shape can obtain more does not mean it is a bad example. It is simple and easy to understand.

Fair enough