

What is the definition of "interface" in object oriented programming

Asked 14 years, 7 months ago Modified 2 years, 2 months ago

Viewed 214k times



A friend of mine goes back and forth on what "interface" means in programming.

159

What is the best description of an "interface"?



To me, an interface is a blueprint of a class. Is this the best definition?



oop

interface

definition

Share

Improve this question

Follow

edited Oct 22, 2022 at 0:36



[maxmitz](#)

180 ● 1 ● 6 ● 21

asked May 19, 2010 at 15:34



[Daniel Kivatinos](#)

25k ● 23 ● 63 ● 81

-
- 1 not a dupe, but it sheds a lot of light on this question:
stackoverflow.com/questions/444245/... – [rmeador](#) May 19, 2010 at 16:24
-

- 1 I would say it is a generic word, but still means the same thing for me. It is like a wall for a particular or a more abstract entity providing some input points to outside and giving them output without any knowledge of internal operations. I think it can be called as an abstraction to define lower to higher level classes in OOP. – [ikbal](#) Oct 7, 2016 at 11:06
-

16 Answers

Sorted by:

Highest score (default)



An interface is one of the more overloaded and confusing terms in development.

257



It is actually a concept of abstraction and encapsulation. For a given "box", it *declares* the "inputs" and "outputs" of that box. In the world of software, that usually means the operations that can be invoked on the box (along with arguments) and in some cases the return types of these operations.



What it does not do is define what the semantics of these operations are, although it is commonplace (and very good practice) to document them in proximity to the declaration (e.g., via comments), or to pick good naming conventions. Nevertheless, there are no guarantees that these intentions would be followed.

Here is an analogy: Take a look at your television when it is off. Its interface are the buttons it has, the various plugs, and the screen. Its semantics and behavior are that it takes inputs (e.g., cable programming) and has outputs (display on the screen, sound, etc.). However,

when you look at a TV that is not plugged in, you are projecting your expected semantics into an interface. For all you know, the TV could just explode when you plug it in. However, based on its "interface" you can assume that it won't make any coffee since it doesn't have a water intake.

In object oriented programming, an interface generally defines the set of methods (or messages) that an instance of a class that has that interface could respond to.

What adds to the confusion is that in some languages, like Java, there is an actual interface with its language specific semantics. In Java, for example, it is a set of method declarations, with no implementation, but an interface also corresponds to a type and obeys various typing rules.

In other languages, like C++, you do not have interfaces. A class itself defines methods, but you could think of the interface of the class as the declarations of the non-private methods. Because of how C++ compiles, you get header files where you could have the "interface" of the class without actual implementation. You could also mimic Java interfaces with abstract classes with pure virtual functions, etc.

An interface is most certainly not a blueprint for a class. A blueprint, by one definition is a "detailed plan of action". An interface promises nothing about an action! The source of the confusion is that in most languages, if you

have an interface type that defines a set of methods, the class that implements it "repeats" the same methods (but provides definition), so the interface looks like a skeleton or an outline of the class.

Share Improve this answer

Follow

edited May 10, 2019 at 20:21



MikeSchinkel

5,036 ● 5 ● 40 ● 48

answered May 19, 2010 at 15:42



Uri

89.6k ● 51 ● 226 ● 322

I am reading a book on Objective-C and it seems to me that the authors use the terms "protocol" and "interface" interchangeably. Is it correct to say that "protocol" and "interface" are the same thing or am I missing something?
– [Fazzolini](#) Jul 4, 2015 at 3:52

-
- 1 I wouldn't use the word "protocol" for "interface." The word "protocol" implies a level of detail about how actions are carried out. "Interface" really is an excellent word with a pure English definition that strictly describes exactly what it is.
– [OCDev](#) Aug 18, 2016 at 11:12

In Windows programming interfaces are used quite widely, so even in C++ you can meet "*interfaces*" in a way of objects that are decoupled from the implementation. – [Victoria](#) Jul 3, 2017 at 18:46 ✎



Consider the following situation:



You are in the middle of a large, empty room, when a zombie suddenly attacks you.

You have no weapon.

Luckily, a fellow living human is standing in the doorway of the room.

"Quick!" you shout at him. "Throw me something I can hit the zombie with!"

Now consider:

You didn't specify (nor do you care) exactly **what** your friend will choose to toss;

...But it doesn't matter, as long as:

- It's something that **can** be tossed (He can't toss you the sofa)
- It's something that you can grab hold of (Let's hope he didn't toss a shuriken)
- It's something you can use to bash the zombie's brains out (That rules out pillows and such)

It doesn't matter whether you get a baseball bat or a hammer -

as long as it implements your three conditions, you're good.

To sum it up:

When you write an interface, you're basically saying: "I need something that..."

Share Improve this answer

answered Jan 9, 2013 at 19:14

Follow



[Yehuda Shapira](#)

8,630 ● 6 ● 45 ● 66

37 Actually, a pillow would still work. You can hit the zombie with it. Bash the zombie's brains out...well that's a question of performance, which is never part of the interface. – [meustrus](#)
Sep 29, 2017 at 4:37



43

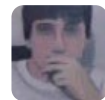


Interface is a contract you should comply to or given to, depending if you are implementer or a user.

Share Improve this answer

answered May 19, 2010 at 15:37

Follow




[Eugene Kuleshov](#)

31.8k ● 5 ● 70 ● 67



1 I actually don't like the unqualified term contract here, because a contract often implies semantics or behavior (as in design by contract). Not all interfaces imply any semantics or behavior. For example, a "hole in the wall" is a real-world interface. Whether you perceive this as a window or as a garbage disposal or as anything is your interpretation. – [Uri](#)
May 19, 2010 at 16:13 ✎

- 4 True. An interface is a "method signature contract", meaning it guarantees to implement the given methods. It makes no guarantee on whether it does so in any given manner.
– [Erik Funkenbusch](#) May 19, 2010 at 17:03
-

Exactly, you should give more explanation to this definition. But an interface is a contract between two parts. Code by contract.... +1 – [matiasnj](#) Aug 24, 2011 at 20:25 

This is a great answer because it's concise and denotative, especially in combination with the comment clarifying that the contract "guarantees to implement the given methods."
Pulling all the best parts of this answer together gives: An interface is like a contract between a software project and a programmer who chooses to contribute to the project by adding an extension class. Upon "signing the contract" the programmer is agreeing that the extension (some subclass of a class) is able to implement a predetermined set of methods (i.e., those of the interface). – [Aether](#) May 7, 2023 at 15:30



25

I don't think "blueprint" is a good word to use. A blueprint tells you how to build something. An interface specifically avoids telling you how to build something.



An interface defines how you can interact with a class, i.e. what methods it supports.



Share Improve this answer

answered May 19, 2010 at 15:38

Follow



[Dave Costa](#)

48.1k ● 8 ● 60 ● 73

2 I believe the requestor asked what is a definition and not what it is not. :) – [Eugene Kuleshov](#) May 19, 2010 at 15:42



10

In Programming, an interface defines what the behavior a an object will have, but it will not actually specify the behavior. It is a contract, that will guarantee, that a certain class can do something.



Consider this piece of C# code here:



```
using System;

public interface IGenerate
{
    int Generate();
}

// Dependencies
public class KnownNumber : IGenerate
{
    public int Generate()
    {
```



```

        return 5;
    }
}

public class SecretNumber : IGenerate
{
    public int Generate()
    {
        return new Random().Next(0, 10);
    }
}

// What you care about
class Game
{
    public Game(IGenerate generator)
    {
        Console.WriteLine(generator.Generate())
    }
}

new Game(new SecretNumber());
new Game(new KnownNumber());

```

The Game class requires a secret number. For the sake of testing it, you would like to inject what will be used as a secret number (this principle is called Inversion of Control).

The game class wants to be "open minded" about what will actually create the random number, therefore it will ask in its constructor for "anything, that has a Generate method".

First, the interface specifies, what operations an object will provide. It just contains what it looks like, but no actual implementation is given. This is just the signature of the method. Conventionally, in C# interfaces are

prefixed with an I. The classes now implement the IGenerate Interface. This means that the compiler will make sure, that they both have a method, that returns an int and is called `Generate`. The game now is being called two different object, each of which implementant the correct interface. Other classes would produce an error upon building the code.

Here I noticed the blueprint analogy you used:

A class is commonly seen as a blueprint for an object. An Interface specifies something that a class will need to do, so one could argue that it indeed is just a blueprint for a class, but since a class does not necessarily need an interface, I would argue that this metaphor is breaking. Think of an interface as a contract. The class that "signs it" will be legally required (enforced by the compiler police), to comply to the terms and conditions in the contract. This means that it will have to do, what is specified in the interface.

This is all due to the statically typed nature of some OO languages, as it is the case with Java or C#. In Python on the other hand, another mechanism is used:

```
import random

# Dependencies
class KnownNumber(object):
    def generate(self):
        return 5

class SecretNumber(object):
    def generate(self):
```

```

        return random.randint(0,10)

# What you care about
class SecretGame(object):
    def __init__(self, number_generator):
        number = number_generator.generate()
        print number

```

Here, none of the classes implement an interface. Python does not care about that, because the `SecretGame` class will just try to call whatever object is passed in. If the object HAS a `generate()` method, everything is fine. If it doesn't: KAPUTT! This mistake will not be seen at compile time, but at runtime, so possibly when your program is already deployed and running. C# would notify you way before you came close to that.

The reason this mechanism is used, naively stated, because in OO languages naturally functions aren't first class citizens. As you can see, `KnownNumber` and `SecretNumber` contain JUST the functions to generate a number. One does not really need the classes at all. In Python, therefore, one could just throw them away and pick the functions on their own:

```

# OO Approach
SecretGame(SecretNumber())
SecretGame(KnownNumber())

# Functional Approach

# Dependencies
class SecretGame(object):
    def __init__(self, generate):
        number = generate()
        print number

```

```
SecretGame(lambda: random.randint(0,10))
SecretGame(lambda: 5)
```

A lambda is just a function, that was declared "in line, as you go". A delegate is just the same in C#:

```
class Game
{
    public Game(Func<int> generate)
    {
        Console.WriteLine(generate())
    }
}

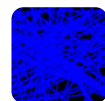
new Game(() => 5);
new Game(() => new Random().Next(0, 10));
```

Side note: The latter examples were not possible like this up to Java 7. There, Interfaces were your only way of specifying this behavior. However, Java 8 introduced lambda expressions so the C# example can be converted to Java very easily (`Func<int>` becomes `java.util.function.IntSupplier` and `=>` becomes `->`).

Share Improve this answer

Follow

edited Mar 23, 2020 at 16:37



iron9

505 ● 4 ● 14

answered Apr 11, 2013 at 8:07



cessor

1,067 ● 11 ● 17



7

To me an interface is a blueprint of a class, is this the best definition?



No. A blueprint typically includes the internals. But a interface is purely about what is visible on the outside of a class ... or more accurately, a family of classes that implement the interface.



The interface consists of the signatures of methods and values of constants, and also a (typically informal) "behavioral contract" between classes that implement the interface and others that use it.

Share Improve this answer

answered May 19, 2010 at 15:39

Follow



Stephen C

718k ● 95 ● 844 ● 1.3k



5

Technically, I would describe an interface as a set of ways (methods, properties, accessors... the vocabulary depends on the language you are using) to interact with an object. If an object supports/implements an interface, then you can use all of the ways specified in the interface to interact with this object.



Semantically, an interface could also contain conventions about what you may or may not do (e.g., the order in which you may call the methods) and about what, in return, you may assume about the state of the object given how you interacted so far.

Share Improve this answer

answered May 19, 2010 at 15:46

Follow



[Ghislain Fourny](#)

7,254 ● 1 ● 31 ● 38



3

Personally I see an interface like a template. If a interface contains the definition for the methods `foo()` and `bar()`, then you know every class which uses this interface has the methods `foo()` and `bar()`.



Share Improve this answer

answered May 19, 2010 at 16:06



Follow



[Thomas Winsnes](#)

1,961 ● 1 ● 13 ● 15





2



Let us consider a Man(User or an Object) wants some work to be done. He will contact a middle man(Interface) who will be having a contract with the companies(real world objects created using implemented classes). Few types of works will be defined by him which companies will implement and give him results. Each and every company will implement the work in its own way but the result will be same. Like this User will get its work done using an single interface. I think Interface will act as visible part of the systems with few commands which will be defined internally by the implementing inner sub systems.

Share Improve this answer

answered Jan 25, 2013 at 13:59

Follow



Arun N

21 ● 2



1



An interface separates out operations on a class from the implementation within. Thus, some implementations may provide for many interfaces.

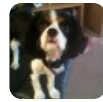
People would usually describe it as a "contract" for what must be available in the methods of the class.

It is absolutely not a blueprint, since that would also determine implementation. A full class definition could be said to be a blueprint.

Share Improve this answer

answered May 19, 2010 at 15:42

Follow



Puppy

147k ● 40 ● 266 ● 477



1



An interface defines what a class that inherits from it must implement. In this way, multiple classes can inherit from an interface, and because of that inheritance, you can

- be sure that all members of the interface are implemented in the derived class (even if its just to throw an exception)
- Abstract away the class itself from the caller (cast an instance of a class to the interface, and interact with it without needing to know what the actual derived class IS)

for more info, see this <http://msdn.microsoft.com/en-us/library/ms173156.aspx>

Share Improve this answer

answered May 19, 2010 at 15:48

Follow



Greg Olmstead

1,561 ● 11 ● 22



1



In my opinion, interface has a broader meaning than the one commonly associated with it in Java. I would define "interface" as a set of available operations with some common functionality, that allow controlling/monitoring a module.

In this definition I try to cover both programatic interfaces, where the client is some module, and human interfaces



(GUI for example).

As others already said, an interface always has some contract behind it, in terms of inputs and outputs. The interface does not promise anything about the "how" of the operations; it only guarantees some properties of the outcome, given the current state, the selected operation and its parameters.

Share Improve this answer

edited May 19, 2010 at 16:14

Follow

answered May 19, 2010 at 15:56



Eyal Schneider

22.4k ● 5 ● 50 ● 78



As above, synonyms of "contract" and "protocol" are appropriate.

1



The interface comprises the methods and properties you can expect to be exposed by a class.



So if a class `Cheetos Bag` implements the `Chip Bag` interface, you should expect a `Cheetos Bag` to behave exactly like any other `Chip Bag`. (That is, expose the `.attemptToOpenWithoutSpillingEverywhere()` method, etc.)

Share Improve this answer

answered May 19, 2010 at 16:36

Follow



wlangstroth

1,070 ● 6 ● 12



1



[A boundary across which two systems communicate.](#)

Interfaces are how some OO languages achieve [ad hoc polymorphism](#). Ad hoc polymorphism is simply functions with the same names operating on different types.



Share Improve this answer

answered Aug 10, 2013 at 7:05

Follow



cheecheeo

672 ● 7 ● 20

I think Golang has this, whereas you won't get name collision having two functions sharing the same name, as long as it has different type ? 🤔 😊 – [projektorius96](#) Feb 25, 2023 at 20:33



1



Conventional Definition - An interface is a contract that specifies the methods which needs to be implemented by the class implementing it.



The Definition of Interface has changed over time. Do you think Interface just have method declarations only ? What about static final variables and what about default definitions after Java 5.



Interfaces were introduced to Java because of the Diamond problem with multiple Inheritance and that's

what they actually intend to do.

Interfaces are the constructs that were created to get away with the multiple inheritance problem and can have abstract methods , default definitions and static final variables.

<http://www.quora.com/Why-does-Java-allow-static-final-variables-in-interfaces-when-they-are-only-intended-to-be-contracts>

Share Improve this answer

answered Aug 25, 2014 at 17:50

Follow



Vivek Vermani

2,004 ● 20 ● 47



0

In short, The basic problem an interface is trying to solve is to separate how we use something from how it is implemented. But you should consider interface [is not a contract](#). Read more [here](#).



Share Improve this answer

answered Oct 12, 2018 at 9:51

Follow



Alireza Rahmani Khalili

2,934 ● 2 ● 33 ● 37



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.