When to use virtual destructors?

Asked 12 years, 5 months ago Active 10 days ago Viewed 752k times

1633

676

I have a solid understanding of most OOP theory but the one thing that confuses me a lot is virtual destructors.

I thought that the destructor always gets called no matter what and for every object in the chain.

When are you meant to make them virtual and why?

c++ polymorphism shared-ptr virtual-destructor

Share Improve this question

Follow

edited Jun 29 '20 at 18:42

asmmo

6,425 1 7 22

asked Jan 20 '09 at 12:58

Lodle

28.2k 19 59 89

6 See this: Virtual Destructor – Naveen Jan 20 '09 at 13:04

159 Every destructor *down* gets called no matter what. `virtual` makes sure it starts at the top instead of the middle. – Mooing Duck Jun 29 '13 at 0:32

15 related question: When should you not use virtual destructors? – Eitan T Aug 4 '13 at 16:39

1 @FranklinYu it's good that you asked because now I can't see any issue with that comment (except trying to give answer in comments). – Euri Pinhollow Nov 8 '17 at 7:02

1 @Nibor: Yes, *if you use that notion*. About half the people I talk to view superclasses as "above", and half view superclasses as "below", so both are conflicting standards, which makes everything confusing. I think superclass as "above" is slightly more common, but that's not the way I was taught :(– Mooing Duck Jun 20 '19 at 17:09

17 Answers

Active | Oldest | Votes

1697

Virtual destructors are useful when you might potentially delete an instance of a derived class through a pointer to base class:

```
class Base
{
    // some virtual methods
};

class Derived : public Base
{
    ~Derived()
    {
        // Do some important cleanup
```

Here, you'll notice that I didn't declare Base's destructor to be `virtual`. Now, let's have a look at the following snippet:

```
Base *b = new Derived();
// use b
delete b; // Here's the problem!
```

Since Base's destructor is not `virtual` and `b` is a `Base*` pointing to a `Derived` object, `delete b` has undefined behaviour:

> [In `delete b`], if the static type of the object to be deleted is different from its dynamic type, the static type shall be a base class of the dynamic type of the object to be deleted and **the static type shall have a virtual destructor or the behavior is undefined**.

In most implementations, the call to the destructor will be resolved like any non-virtual code, meaning that the destructor of the base class will be called but not the one of the derived class, resulting in a resources leak.

To sum up, always make base classes' destructors `virtual` when they're meant to be manipulated polymorphically.

If you want to prevent the deletion of an instance through a base class pointer, you can make the base class destructor protected and nonvirtual; by doing so, the compiler won't let you call `delete` on a base class pointer.

You can learn more about virtuality and virtual base class destructor in this article from Herb Sutter.

Share Improve this answer

Follow

edited Oct 23 '18 at 18:57

einpoklum

88.5k 41 231 459

answered Jan 20 '09 at 13:04

Luc Touraille

73.3k 15 82 134

211 This would explain why i had massive leaks using a factory i made before. All makes sense now. Thanks – Lodle Jan 20 '09 at 13:08

11 Well, this is a bad example as there are no data members. What if `Base` and `Derived` have *all* automatic storage variables? ie there is no "special" or additional custom code to execute in the destructor. Is it ok then to leave off writing any destructors at all? Or will the derived class *still* have a memory leak? – bobobobo Jul 8 '12 at 18:27

4 Wait, it will be undefined behavior – bobobobo Jul 8 '12 at 18:29

37 From the Herb Sutter's article: "Guideline #4: A base class destructor should be either public and virtual, or protected and nonvirtual." – Sundae Feb 9 '16 at 8:22

4 Also from the article - 'if you delete polymorphically without a virtual destructor, you summon the dreaded specter of "undefined behavior," a specter I personally would rather not meet in even a moderately well-lit alley, thank you very much.' lol – Bondolin Feb 29 '16 at 14:30