FAQs Search Recent Topics Flagged Topics Hot Topics Best Topics Register / Login

Pie is fun! So are jokes, and recipes, and lots of other cool stuff!
Visit the Meaningless Drivel forum and join the fun!

Win a copy of Hands On Software Engineering with Python this week in the Jython/Python forum!

Bookmark Topic Watch Topic

programming forums

Java

Mobile

Certification

Databases

Caching

Books

Engineering

Micro Controllers

OS

Languages

Paradigms

IDEs

Build Tools

Frameworks

Application Servers

Open Source

This Site

Careers

Other

all forums

this forum made possible by our

Forum: Wiki

WIKI Overriding Vs Hiding

posted 3 years ago

2 +1 Report

Can I override a static method?

Many people have heard that you can't override a static method. This is true - you can't. However it is possible to write code like this:

```
class Foo {
    public static void method() {
        System.out.println("in Foo");
    }
}

class Bar extends Foo {
    public static void method() {
        System.out.println("in Bar");
    }
}
```

This compiles and runs just fine. Isn't it an example of a static method overriding another static method? The answer is no - it's an example of a static method *hiding* another static method. If you try to override a static method, the compiler doesn't actually stop you - it just doesn't do what you think it does.

So what's the difference?

Briefly, when you *override* a method, you still get the benefits of run-time polymorphism, and when you *hide*, you don't. So what does that mean? Take a look at this code:

```
class Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Foo");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Foo");
    }
}

class Bar extends Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Bar");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Bar");
    }
}

class Test {
    public static void main(String[] args) {
        Foo f = new Bar();
        f.instanceMethod();
        f.classMethod();
    }
}
```

If you run this, the output is

```
instanceMethod() in Bar
classMethod() in Foo
```

Why do we get instanceMethod from Bar, but classMethod() from Foo? Aren't we using the same instance f to access both of these? Yes we are - but since one is overriding and the other is hiding, we see different behavior.

Since instanceMethod() is (drum roll please...) an instance method, in which Bar *overrides* the method from Foo, at run time the JVM uses the *actual* class of the instance f to determine which method to run. Although f was *declared* as a Foo, the actual instance we created was a new Bar(). So at runtime, the JVM finds that f is a Bar instance, and so it calls instanceMethod() in Bar rather than the one in Foo. That's how Java normally works for instance methods.

With classMethod() though. since (*ahem*) it's a *class* method, the compiler and JVM don't expect to need an actual instance to invoke the method. And even if you provide one (which we did: the instance referred to by f) the JVM will never look at it. The compiler will only look at the *declared type* of the reference, and use that declared type to determine, *at compile time*, which method to call. Since f is declared as type Foo, the compiler looks at f.classMethod() and decides it means Foo.classMethod. It doesn't matter that the instance reffered to by f is actually a Bar - for static methods, the compiler only uses the declared type of the reference. That's what we mean when we say a static method does not have run-time polymorphism.

Because instance methods and class methods have this important difference in behavior, we use different terms - "overriding" for instance methods and "hiding" for class methods - to distinguish between the two cases. And when we say you can't override a static method, what that means is that even if you write code that *looks* like it's overriding a static method (like the first Foo and Bar at the top of this page) - it won't *behave* like an overridden method.

So what about accessing a static method using an instance?

It's possible in Java to write something like:

```
f.classMethod();
```

where f is an instance of some class, and classMethod() is a class method (i.e. a static method) of that class. This is legal, but it's a bad idea because it creates confusion. The actual *instance* f is not really important here. Only the *declared type* of f matters. That is, what class is f declared to be? Since classMethod() is static, the class of f (as determined by the compiler at compile time) is all we need.

Rather than writing:

```
f.classMethod();
```

It would be better coding style to write either:

```
Foo.classMethod();
```

or

```
Bar.classMethod();
```

That way, it is crystal clear which class method you would like to call. It is also clear that the method you are calling is indeed a class method.

Barring that, you could always come up with this monstrosity:

```
f.getClass().getMethod("classMethod", new Class[]).invoke(null, new
```

But all this could be avoided by simply not trying to override your static (class) methods. :-)

Why does the compiler sometimes talk about overriding static methods?

Sometimes you will see error messages from the compiler that talk about overriding static methods. Apparently, whoever writes these particular messages has not read the Java Language Specification and does not know the difference between overriding and hiding. So they use incorrect and misleading terminology. Just ignore it. The Java Language Specification is very clear about the difference between overriding and hiding, even if the compiler messages are not. Just pretend that the compiler said "hide" rather than "override"..

and POOF! You're gone! But look, this tiny ad is still here:

RavenDB is an Open Source NoSQL Database that’s fully transactional (ACID) across your database
https://coderanch.com/t/704633/RavenDB-Open-Source-NoSQL-Database

Bookmark Topic Watch Topic New Topic

Boost this thread!

Similar Threads

Why it is illegal to use abstract and static modifiers together in method declaration ?

overriding static method

Polymorphism and Static methods

Static methods

what is the difference between redefining and overridin a method in a subclass

More...

current ranch time (not your local time) is **Jan 18, 2019 03:43:29**