**Why below has compile time error**

**public** **class** ListJdk4 {

**public** **static** **void** main(String[] args) {

}

**public** **void** add(List<String> s){

}

**public** **void** add(List<Integer> s){

}

}

**Because java erases type at compile time.**

[Method has the same erasure as another method in type](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type)

class Test{

void add(Set<Integer> ii){}

void add(Set<String> ss){}

}

I get the compilation error

Method add(Set) has the same erasure add(Set) as another method in type Test.

while I can work around it, I was wondering why javac doesn't like this.

I can see that in many cases, the logic of those two methods would be very similar and could be replaced by a single

public void add(Set<?> set){}

method, but this is not always the case.

This is extra annoying if you want to have two constructors that takes those arguments because then you can't just change the name of one of the constructors.

[java](https://stackoverflow.com/questions/tagged/java) [generics](https://stackoverflow.com/questions/tagged/generics)

[share](https://stackoverflow.com/q/1998544)[improve this question](https://stackoverflow.com/posts/1998544/edit)

[edited May 30 '15 at 20:38](https://stackoverflow.com/posts/1998544/revisions)

[[https://i.stack.imgur.com/MRt7J.jpg?s=32&g=1](https://stackoverflow.com/users/2694480/mithun)](https://stackoverflow.com/users/2694480/mithun)

[Mithun](https://stackoverflow.com/users/2694480/mithun)

**1,811**31225

asked Jan 4 '10 at 9:58

[[https://www.gravatar.com/avatar/8a9c16d8d4a4295bf2d57ba1094a3fd8?s=32&d=identicon&r=PG](https://stackoverflow.com/users/1930838/omry-yadan)](https://stackoverflow.com/users/1930838/omry-yadan)

[Omry Yadan](https://stackoverflow.com/users/1930838/omry-yadan)

**16.4k**104168

* you can differ by implementation like one is Set other take HashSet etc – [AZ\_](https://stackoverflow.com/users/185022/az) [Mar 30 '14 at 12:55](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment34666757_1998544)
* 1

what if you run out of data structures and you still need more versions? – [Omry Yadan](https://stackoverflow.com/users/1930838/omry-yadan" \o "16,390 reputation) [Apr 9 '14 at 4:04](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment35042884_1998544)

* 1

You could make custom classes that inherit from base versions. – [willlma](https://stackoverflow.com/users/1720014/willlma" \o "4,574 reputation) [Oct 20 '14 at 15:19](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment41575919_1998544)

* 1

OP, did you come up with some solution to the constructor problem? I need to accept two kinds of List and I don't know how to handle it. – [Tomáš Zato](https://stackoverflow.com/users/607407/tom%c3%a1%c5%a1-zato" \o "20,500 reputation) [Apr 12 '15 at 22:55](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment47338029_1998544)

add a comment

6 Answers

[active](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type?answertab=active#tab-top)[oldest](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type?answertab=oldest#tab-top)[votes](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type?answertab=votes#tab-top)

up vote294down voteaccepted

This rule is intended to avoid conflicts in legacy code that still uses raw types.

Here's an illustration of why this was not allowed, [drawn from the JLS.](https://docs.oracle.com/javase/specs/jls/se8/html/jls-8.html#jls-8.4.2) Suppose, before generics were introduced to Java, I wrote some code like this:

class CollectionConverter {

List toList(Collection c) {...}

}

You extend my class, like this:

class Overrider extends CollectionConverter{

List toList(Collection c) {...}

}

After the introduction of generics, I decided to update my library.

class CollectionConverter {

<T> List<T> toList(Collection<T> c) {...}

}

You aren't ready to make any updates, so you leave your Overrider class alone. In order to correctly override the toList() method, the language designers decided that a raw type was "override-equivalent" to any generified type. This means that although your method signature is no longer formally equal to my superclass' signature, your method still overrides.

Now, time passes and you decide you are ready to update your class. But you screw up a little, and instead of editing the existing, raw toList() method, you *add* a new method like this:

class Overrider extends CollectionConverter {

@Override

List toList(Collection c) {...}

@Override

<T> List<T> toList(Collection<T> c) {...}

}

Because of the override equivalence of raw types, both methods are in a valid form to override the toList(Collection<T>) method. But of course, the compiler needs to resolve a single method. To eliminate this ambiguity, classes are not allowed to have multiple methods that are override-equivalent—that is, multiple methods with the same parameter types after erasure.

The key is that this is a language rule designed to maintain compatibility with old code using raw types. It is not a limitation required by the erasure of type parameters; because method resolution occurs at compile-time, adding generic types to the method identifier would have been sufficient.

[share](https://stackoverflow.com/a/8467804)[improve this answer](https://stackoverflow.com/posts/8467804/edit)

[edited Nov 12 '17 at 23:21](https://stackoverflow.com/posts/8467804/revisions)

answered Dec 11 '11 at 21:53

[[https://www.gravatar.com/avatar/ffbf4e85b8ffbae4e9039b9c0cf07bc8?s=32&d=identicon&r=PG](https://stackoverflow.com/users/3474/erickson)](https://stackoverflow.com/users/3474/erickson)

[erickson](https://stackoverflow.com/users/3474/erickson)

**214k**42321418

* 2

Great answer and example! I am not sure, however, if I fully understand your last sentence ("Because method resolution occurs at compile-time, before erasure, type reification is not required to make this work."). Could you elaborate a bit? – [Jonas Eicher](https://stackoverflow.com/users/1412324/jonas-eicher) [Dec 4 '12 at 13:56](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment18819961_8467804)

* 2

Makes sense. I just spent some time thinking about type reification in template methods, but yeah: the compiler makes sure the right method gets selected before type erasure. Beautiful. If it weren't tainted by the legacy code compatibility issues. – [Jonas Eicher](https://stackoverflow.com/users/1412324/jonas-eicher) [Dec 5 '12 at 9:03](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment18845270_8467804)

* 1

@daveloyall No, I am not aware of such an option for javac. – [erickson](https://stackoverflow.com/users/3474/erickson" \o "214,053 reputation) [Dec 4 '14 at 0:32](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment43036552_8467804)

* 6

Not the first time I encounter Java error which is no error at all and could be compiled if only the authors of Java used warnings as everybody else does. Only they think they know everything better. – [Tomáš Zato](https://stackoverflow.com/users/607407/tom%c3%a1%c5%a1-zato" \o "20,500 reputation) [Apr 12 '15 at 22:50](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment47337940_8467804)

* 1

@Andrey Fixed now, thanks for letting me know – [erickson](https://stackoverflow.com/users/3474/erickson" \o "214,053 reputation) [Mar 19 '17 at 16:52](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type#comment72880377_8467804)

[show **10** more comments](https://stackoverflow.com/questions/1998544/method-has-the-same-erasure-as-another-method-in-type)

up vote93down vote

Java generics uses type erasure. The bit in the angle brackets (<Integer> and <String>) gets removed, so you'd end up with two methods that have an identical signature (the add(Set) you see in the error). That's not allowed because the runtime wouldn't know which to use for each case.

If Java ever gets reified generics, then you could do this, but that's probably unlikely now.

v