Type Erasure and Overloading in Java: Why does this work?

Asked 13 years, 8 months ago Modified 13 years, 8 months ago Viewed 3k times

19

I have the following code:

```
public class Pair< T, U > {
    public T first;
    public U second;
}
public class Test {
    public int method( Pair< Integer, Integer > pair ) {
        return 0;
    }
    public double method( Pair< Double, Double > pair ) {
        return 1.0;
    }
}
```

This actually compiles and works like one would expect. But if the return types are made to be the same, this doesn't compile, with the expected "name clash: method(Pair) and method(Pair) have the same erasure"

Given that the return type isn't part of the method signature, how is this overloading possible?

java overloading type-erasure

Share Improve this question Follow

asked Apr 3, 2011 at 2:44
Kyle Dewey
690 ●5 ●8

1 possible duplicate of Java generics code compiles with javac, fails with Eclipse Helios
– BalusC Apr 3, 2011 at 3:18

4 Answers

Sorted by: Highest score (default)

9

Consider the following 4 methods

```
            Java code                        bytecode
```

```
m1:    Byte f(List<Byte> list)           f List -> Byte
m2:    Long f(List<Byte> list)           f List -> Long
m3:    Byte f(List<Long> list)           f List -> Byte
m4:    Long f(List<Long> list)           f List -> Long
```

According to the current Java Language Spec,

- m1 and m2 cannot coexist, nor can m3 and m4. because they have the same parameter types.
- m1 and m3 can coexist, so can m1 and m4. because they have different parameter types.

But javac 6 only allows m1+m4, not m1+m3. That's related to the bytecode representation of methods, which includes return types. Therefore, m1+m4 are ok, but not m1+m3.

This is a screwup where Java and JVM specs don't see eye to eye. There is no "correct" way for javac.

While it sucks, the good news is, overloading is a vanity, not a necessity. We can always use different, more descriptive and distinct names for these methods.

Share
Improve this answer
Follow

edited Apr 3, 2011 at 10:09

answered Apr 3, 2011 at 9:47
irreputable
45.4k ● 9 ● 68 ● 93

2 According to what the Spec should say (according to several entries in Java's bug database), no two of these methods can coexist, because the have the same parameter types **after type erasure**. – Christian Semrau Apr 3, 2011 at 10:30

Thanks for the bytecode representation; that makes the problem very clear. So even though "method signature" typically refers to just the parameters, a method signature in bytecode includes the return type. – Kyle Dewey Apr 3, 2011 at 16:02

And according to this bug even m1+m4 would be illegal in javac7. It's a pity as sometimes it's very usable to have few overloaded methods in form of `SomeType f(SomeClass<SomeType>)` . (Like in my answer to this question) – Volo Jul 13, 2011 at 14:19

6

Overloading is done at compile-time.

Although the generic parameters are erased at run-time, they're still available to the compiler to resolve overloads.

Share Improve this answer Follow

answered Apr 3, 2011 at 2:57

SLaks
886k ● 181 ● 1.9k ● 2k

3

A Java method signature actually does include the return type; if you've ever worked with JNI you've seen type descriptors like *(LPair;)D* and *(LPair;)I*. That last character denotes the return types of your two methods. Although the Java language rule is that the parameters must differ for two overloaded methods, the class file format can actually distinguish methods based only on their return types. When there is generic type information to allow the compiler to resolve the overloads based on their arguments, then as long as the return types are different the erasures will have different signatures, and it all works fine. If the returns types are the same, though, then after erasure the methods have the same signature, the class file can't tell the difference, and you have a problem.

Share Improve this answer Follow

answered Apr 3, 2011 at 3:03
Ernest Friedman-Hill
81.6k ● 11 ● 157 ● 190

1 This is the best explanation I could find on this anywhere. Thanks – patentfox Jun 6, 2017 at 7:01

The return type is included in the method descriptor, but not the signature. I am not sure if this difference between "descriptor" and "signature" is Java-specific, however. Source: angelikalanger.com/GenericsFAQ/FAQSections/…? – Chthonic Project Oct 1, 2019 at 18:58

-1

I believe that you are actually looking at typing wrong. By saying that the first method takes Pair you are giving it a very specific type. It's like saying method(String, String). The second method of Pair is like saying method(Person, Person). This too is very specific at a typing level. If you would change your methods to be method(Pair`<T,U>`, Pair`<T,U>`) and have that twice, you'd break the compile.

So short answer: since you've strongly typed Pair to mean two different things, Generics aren't at play, just typing rule.

Share Improve this answer Follow

answered Apr 3, 2011 at 2:57
Virmundi
2,631 ● 4 ● 27 ● 36