

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2008-04/msg00647.html>

- *From:* Lew <lew@xxxxxxxxxxxxxx>
 - *Date:* Mon, 07 Apr 2008 08:31:53 -0400
-

Zig wrote:

On Sun, 06 Apr 2008 10:15:12 -0400, Lew <lew@xxxxxxxxxxxxxx> wrote:

You proposed that to be "compatible" with pre-1.0 days, Hashtable and Dictionary would have to be implemented as having <Object, Object> parameters.

I made no such proposal: I merely followed your earlier line of reasoning to an outcome that was inconsistent.

You said,

If your speculation was correct, then java.util.Hashtable and java.util.Dictionary would also be required to be of type <Object, Object>.

I pointed out that that is not correct, because Hashtable and Dictionary are parametrized and Properties is not. therefore there is no inconsistency, because the situations are not comparable.

At most it would have affected compile-time compatibility.

Yes, that is the entire point. What is this "at most" stuff? That **is** the point.

I admit this could be the reason. But it seems highly unlikely. Sun has **never** guaranteed compile-time compatibility, and treats it only as a nicety. If we return to the example:

It's sort of useless to speculate why Sun did that, since everyone over there admits it was a mistake for

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Properties to inherit from Hashtable. I was simply pointing out technical requirements that would flow from that original misstep. I speculate that if Sun had thought those consequences through, they'd have done it differently.

```
Object o="foo";  
new Properties().put(o, "bar");
```

If properties were of type <String,String>, then this would generate a compile time warning, letting the programmer know they've done something in a way they are not supposed to. The source fix is completely trivial: change the first line to:

```
String o="foo";
```

Given the simplicity of the code fix, plus the ability to compile using "-source 1.4" until the source can be corrected, I would bet that Sun would consider the benefit of additional error checking to outweigh the inconvenience of source changes during the developer's transition to Java 5. This would also have made the put & putAll methods generate warnings when adding non-Strings, re-enforcing the discouraged practice of putting raw objects into these methods, which would have been a good thing.

Sun didn't want to force people to use that flag. What if you wanted to use Properties and, say, enums in the same code?

As for the benefit of additional error checking, that would not be present in "-source 1.4" since you'd suppress the use of generics that way.

Leaving no choice but <Object, Object>. That *is* the technical equivalent of the pre-generic class.

Actually, you show a way to abuse Properties. You aren't supposed to use 'put()':

the put and putAll methods can be applied to a Properties object. Their use is strongly discouraged

You are spot-on here. As I was thinking about the more likely reason for Sun to do this, I came across something to point out.

Everyone already knows why Sun did this – they made a mistake deriving Properties from Hashtable way back, then the introduction of generics forced them to derive from the closest equivalent parametrization of Hashtable, that being <Object, Object>.

Properties has a subclass within the JDK: java.security.Provider. But look at the example usage of java.security.Provider:

...

Oooooops. Sun violated their own convention, and used put instead of setProperty. Shame on

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

them: I should probably file that.

You have a gift for using Sun's mistakes and trying to prove points based on them. Provider has the same lack of relationship to Properties as Properties does to Hashtable. You cannot make conclusions about how to use Provider based on Properties' documentation, and you cannot make conclusions about Properties based on Hashtable's documentation.

Let's say a Java 1.4 developer copied the above code snippet. They may have carefully read the documentation for the Provider class, but not realized that the documentation in Properties makes use of non-string elements discouraged. In their initialization block, they simply write:
`put("Signature.SHA1withDSA KeySize", new Integer(1024));`

Irrelevant. Besides, they wouldn't even use Provider, since the documentation there tells us:

The service type Provider is reserved for use by the security framework.

We're not supposed to use it. So once again you are trying to make a point with an idiom that is either forbidden or strongly discouraged in Java.

They hook in their provider, write their application, and since they don't use the load/store methods, everything runs just fine & dandy.

Again from the Provider Javadocs:

Services of this type cannot be added, removed, or modified by applications.

Please read the Javadocs.

Now suppose Sun updates Properties to be of type <String,String>. The

Never going to happen. It's not the matching type to the raw type, i.e., the only type available prior to generics.

Anyway, it's obvious that one "put" method performs either up-casting or down-casting, and then acts as a bridge to the other. But, I can't find

No casting involved – generics are compile-time only.

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Key point: This implies that it is valid for javac to create an implementation of put(Object,Object) that casts its [sic] parameters to Strings, and then acts as a bridge to put(String,String).

Generic types are not reified – so that cannot happen. javac cannot create either an <Object, Object> or a <String, String> version, and the parameters are never cast – the raw type accepts Objects only.

Since the Provider class does override put, if Properties were changed to <String,String>, then the put method in Provider would have to change signature to put(String,String) – I think this changes my "key point" from a possibility to a probability.

It's impossible for the reasons stated – there is no run-time parametric typing nor is casting involved, so your "key point" is never going to happen.

Note that if Properties was declared <String, String>, it would break Provider, which is not.

Now, Sun then publishes the JRE, and Properties.put(Object,Object) now casts its parameters to Strings, and in turn calls put(String,String), since this is now legal.

Nope – no casting involved. The arguments to put are always Objects. In HashMap, too.

Our end user then upgrades their JRE. Whoops. Suddenly, their app written by the Java 1.4 developer now fails: it throws some ClassCastExceptions and never runs. The end user switches back to Java 1.4 and everything runs fine again – maybe they can get an upgraded version from their developer, but they could just chalk it up as a Java bug.

You just presented an argument against making Properties inherit Hashtable <String, String>.

Now this outcome strikes me as just the kind of deadly scenario Sun rigorously tries to avoid. Since Sun does try to ensure runtime compatibility such that a compiled binary will run on a newer JVM, this condition would probably constitute a Java bug.

Except that it would be compile-time only. Luckily, Properties inherits Hashtable <Object, Object>, so that scenario won't occur at compile time, either.

So, please allow me to summarize the reasons that we've come up with so far:

* potential for runtime incompatibility when users put non-String entries into a Properties

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>

Nope. No run-time incompatibility issues either way. Since users aren't even supposed to put non-String objects in Properties, that could not have been a factor in the decision to make Properties inherit the <Object, Object> form.

- * compile-time incompatibility when users use the discouraged methods put and putAll, even to store Strings

whilst breaking the formal equivalence and the Provider class. Currently there is no compile-time incompatibility.

- * pre-generics classes may be generified with an unbounded type, or with the bounded type Object, and not with any other bounded type

True.

- * making Properties expose Map methods was a mistake, which Sun is attempting to discourage
- * Sun dropped the ball

Also true.

I've ranked those in order of what strikes me as most probable reasons for Sun to choose to make Properties extend <Object, Object>.

None of them. They did it because that is the only formally equivalent type to the raw Hashtable type.

If I have mis-interpreted your earlier posts, or if you have any other thoughts, I trust you'll respond!

I don't know if you misinterpreted my posts, but you've misinterpreted how generics work, since they don't pertain to run time, and how to use Properties and Provider.

--
Lew

.

Re: java.util.Properties extending from HashMap<Object, Object> instead of HashMap<String, String>