

Plugins that generate serialVersionUID...



diji 1 posts since

Aug 18, 2008

Using 2 different plugins that generate a serialVersionUID field in Java both cause an Exception in IDEA.

Using GenerateSerialVersionUID causes:

Error during dispatching of

```
java.awt.event.InvocationEvent[INVOCATION_DEFAULT,runnable=com.intellij.ui.popup.PopupFactoryImpl$ActionPopupStep$1 @ecfd60,notifier=null,catchExceptions=false,when=1219066113745]
on sun.awt.windows.WToolkit@189e60f: com.intellij.psi.PsiManager.findClass(Ljava/lang/
String;Lcom/intellij/psi/search/GlobalSearchScope;)Lcom/intellij/psi/PsiClass;
```

```
java.lang.NoSuchMethodError: com.intellij.psi.PsiManager.findClass(Ljava/lang/String;Lcom/
intellij/psi/search/GlobalSearchScope;)Lcom/intellij/psi/PsiClass;
```

```
at com.siyeh.ig.psiutils.SerializationUtils.isSerializable(SerializationUtils.java:39)
```

Using SerializeMe causes:

```
com.intellij.psi.PsiManager.getElementFactory()Lcom/intellij/psi/PsiElementFactory;
```

```
java.lang.NoSuchMethodError: com.intellij.psi.PsiManager.getElementFactory()Lcom/intellij/
psi/PsiElementFactory;
```

```
at com.intellij.plugin.serialize.util.PsiFieldHelper.addField(PsiFieldHelper.java:15)
```

Plugins that generate serialVersionUID...

--

OS: Windows XP

JDK: JDK 1.5.0

Tags: plugin, idea, exception



[constv](#) 138 posts since

Nov 8, 2002 1. Re: **Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 20, 2008 8:56 PM

May someone please correct me if I am wrong, but I have always doubted the usefulness of such plug-ins, as well as Sun's *serialver* utility. Here's my point...

If you include *serialVersionUID* in the class data, the serialization mechanism will understand which *past version* of the class you want the current implementation to be compatible with. Normally, if the version IDs are the same the serialization mechanism will be able to apply the correct/acceptable conversion to a different version of the class by ignoring the unexpected (new) fields and using the default values for the missing expected ones. However, such conversion will not be applied if the serial version IDs are not the same, which will always be the case if serialVersionUID is not specified in your class and the class changes, no matter how insignificant the change is. The JVM will generate one for each serializable class based on its content, and if the class changes, the new ID will be different. That's why it is recommended to always provide the serialVersionUID value explicitly.

Now, if you change/refactor the class significantly, you might not want the new version to be considered compatible with the past versions - for obvious reasons. This means, you need to come up with a *new* serialVersionUID. I wonder how many people actually ever do that. Most people probably never change the value of serialVersionUID after they first have generated it. One reason - my guess - is because the generated value is simply unreadable and once it's there, no one even pays attention to it any more. Keeping the value unchanged tells the serialization mechanism that no matter how different the class versions might be, they still should be considered compatible. Is this exactly what you want? So, if you want to ensure that a certain version of your class gets indeed distinguished, you need to update the serialVersionUID value. Cryptic 20-character long values just look scary and unrecognizable. You might as well do the following:

Plugins that generate serialVersionUID...

```
private static final long serialVersionUID = 1L;
```

The effect is just the same, but your code is much cleaner. It is also easier to keep track of your versions and determine when it's time to increment it.

I wonder what other people think.

Thanks,

Constantine



[Mark Vedder](#) 556 posts since

Dec 10, 2003 2. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 21, 2008 6:17 AM

IMHO, just because developers forget to update a field value does not make the mechanism useless and therefore something that should not be used. I know developers that don't catch runtime exceptions, thus allowing an application to die a horrible death. To me, that means the developer failed; not the RuntimeException mechanism. Same holds for the serialVersionUID field.

What would be nice is an inspection that warns the serialVersionUID field needs updating because changes have been made. Or another option, integrate into the version control API and have it automatically insert or update the serialVersionUID field on a commit of any class the implements Serializable. (It would be of course an option that can be turned off like the "Optimize imports" option.)



[constv](#) 138 posts since

Nov 8, 2002 3. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 21, 2008 6:37 PM

Mark Vedder wrote:

What would be nice is an inspection that warns the serialVersionUID field needs updating because changes have been made. Or another option, integrate into the version control

Plugins that generate serialVersionUID...

API and have it automatically insert or update the serialVersionUID field on a commit of any class the implements Serializable. (It would be of course an option that can be turned off like the "Optimize imports" option.)

This is actually a great idea. I think it should be easy to implement since IntelliJ maintains local history and can easily detect if the class has changed. The somewhat tricky part would be to implement this feature in a way that does not become annoying to the developer. You don't want the IDE to prompt a message "Class changes detected. Would you like to update the serialVersionUID value?" each time you type a new character. But I am sure something reasonable/configurable can be done. JetBrains are good at things like that. This indeed would be a really useful intention that would also educate a lot of developers on the proper usage of serialVersionUID.

You probably don't want to have the ID updated automatically on every single class change. This will basically duplicate the default behavior of the VM when the ID is not explicitly provided. The purpose of specifying its value in the class is actually to ensure that different versions of the same class are recognized as compatible! You only need to change the ID when you want to enforce invalidation of the past versions. If you don't provide the ID explicitly, the VM applies such invalidation each time it detects any change within the class. In other words, if you want every single version to be treated as a distinct version, do not provide the serialVersionUID value. But when you do provide it, you have control over when to invalidate the previous versions.

Edited by: constv on Aug 21, 2008 10:31 AM



[Mark Vedder](#) 556 posts since

Dec 10, 2003 4. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 21, 2008 6:01 AM

The built in intention fix works. To use, go into your inspection settings. Under the "Serialization Issues" turn on the "Serializable class without 'serialVersionUID'" inspection. If a serializable class does not have a serialVersionUID, the class name will be highlighted with a warning. Move the cursor there and type alt+Enter to get the intention fix of "Add 'serialVersionUID' field".

Plugins that generate serialVersionUID...

If you need to update the value after making changes to the class, you'll need to delete the 'serialVersionUID' field, then reuse the intention fix.



[constv](#) 138 posts since

Nov 8, 2002 5. Re: **Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 21, 2008 6:20 PM

Mark,

I totally agree that the serial ID must be used - and properly maintained, and I that's what I stated in my previous post. My point, however, is that you don't need any special algorithm to generate such IDs. You can start with 1L, and then increment it each time you introduce updates to your class that should render the past versions obsolete. The serialVersionUID generators, such as the Sun utility and most plug-ins, basically mimic the JVM's algorithms that are applied to a serialized class when the ID is not explicitly specified. In such cases, the JVM must ensure that two distinct versions of the same class (however small the distinction is) always produce distinct IDs. That is why such algorithms basically hash the complete content of the class, so that if even a single character in a field name is changed, the algorithm is (all but) guaranteed to produce a different result.

When you explicitly define the ID value as the class's private static field, it **does not have to be a hash/checksum/whatever-based number**, it may be a simple readable long integer, such as 1L, 2L, 3L, etc - that is *unique for this version of the class* - compared to other versions - as long as you want to render all past versions obsolete. If your changes are insignificant and you still want the version to be treated as compatible with the past version(s), you leave the value unchanged.

It's amazing how many people "program" (or should I say, insert code) without questioning why certain things are necessary and how they should be used. Lots of people will say that serialVersionUID is absolutely necessary in a serialized class (just because Josh Bloch, or someone else, said so) but then they insert it in the code and never maintain it, which completely defeats the purpose and makes the default behavior of the VM in the case of the missing ID actually safer!

Off topic...As far as RT exceptions go, man, you are preaching to the choir! I think the reason so many (most?) Java developers are clueless when it comes to proper error

Plugins that generate serialVersionUID...

handling is the *disastrous* concept of checked exceptions that, sadly, still exists in Java. (Although even Sun is quietly backing out of it w/o the admission of guilt - by not putting checked exceptions in EJB 3.x, and by making most of the new APIs throw RTEs only.) Checked exceptions defeat the original purpose of the exceptions principle. They do not allow the call stack to freely unwind all the way to the only centralized handler that is designed to handle the particular type of error. Instead, they act as glorified error codes that force each caller to check - and, most often, to mishandle the error *before* it gets to the proper handler. As the result, developers don't even bother to design such proper centralized handlers, and error handling becomes a knee-jerk reaction to compiler errors instead of a thoughtful design according to the application requirements. It is scattered all over the place and makes no sense, stack traces are lost, valuable error info is discarded, etc. You are absolutely correct: checked exceptions by definition represent only *sub-sets* of all errors, and just handling checked exceptions is never enough. But so many people never even bother to think about that. Instead they blame RTEs for their own incompetence.

Cheers,

C



[Mark Vedder](#) 556 posts since

Dec 10, 2003 6. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 21, 2008 8:39 PM

constv wrote:

My point, however, is that you don't need any special algorithm to generate such IDs. You can start with 1L, and then increment it each time you introduce updates to your class that should render the past versions obsolete. ... **1L, 2L, 3L**, etc - that is *unique for this version of the class* - compared to other versions

Ah, gotcha... I missed your comment about incrementing in your original post. Sorry about that.

As silly as it sounds, for me, I've always seen the use of a standard generator having a side benefit that I don't have to think about (i.e. remember 😊) whether the changes I made have made the object incompatible with older versions. I just regenerate the ID. Either the same

Plugins that generate serialVersionUID...

number gets generated (compatible) or a new one does. But that's probably me being lazy



constv wrote:

It's amazing how many people "program" (or should I say, insert code) without questioning why certain things are necessary and how they should be used.

How true. I once worked with a guy like this. I called him a "CGI programmer" CGI = Copies (from) Google Inquires.

constv wrote:

Off topic...As far as RT exceptions go, man, you are preaching to the choir! I think the reason so many (most?) Java developers are clueless when it comes to proper error handling is the *disastrous* concept of checked exceptions that, sadly, still exists in Java. (Although even Sun is quietly backing out of it w/o the admission of guilt - by not putting checked exceptions in EJB 3.x, and by making most of the new APIs throw RTEs only.) Checked exceptions defeat the original purpose of the exceptions principle. They do not allow the call stack to freely unwind all the way to the only centralized handler that is designed to handle the particular type of error. Instead, they act as glorified error codes that force each caller to check - and, most often, to mishandle the error *before* it gets to the proper handler. As the result, developers don't even bother to design such proper centralized handlers, and error handling becomes a knee-jerk reaction to compiler errors instead of a thoughtful design according to the application requirements. It is scattered all over the place and makes no sense, stack traces are lost, valuable error info is discarded, etc. You are absolutely correct: checked exceptions by definition represent only *sub-sets* of all errors, and just handling checked exceptions is never enough. But so many people never even bother to think about that. Instead they blame RTEs for their own incompetence.

Some good points. My "favorite" thing is this:

Plugins that generate serialVersionUID...

```
catch (SomeVeryExplicitException e)
{
    throw new RuntimeException("Exception occurred");
}
```

which really means...

```
catch (SomeVeryExplicitException e)
{
    throw new RuntimeExceptionException("Hi, I caught a nice meaningful specific exception with a stack trace. " +
        "But now I am just throwing a general Exception with no useful message other " +
        "than 'Exception occurred'. And I'm not adding the original exception as a " +
        "cause... I'm not even logging it. So you lose your stack trace and you won't " +
        "be able to track down the root issue. Was it a FileNotFoundException? Was " +
        "it an SQLException? Was it a SomeVeryExplicitException? You'll never know. " +
        "But I can sleep at night because I have exception handling in my code. " +
        "I don't have to be bothered doing anything further. I'm not even going to declare " +
        "this Runtime exception in my method signature. And since I have no Javadoc, you " +
        "won't know this is a disaster waiting to happen. Well, at least not until Saturday night " +
        "at 2:00am when you're on call. Have a nice day");
}
```

IMHO opinion, to your comment of : *"I think the reason so many (most?) Java developers are clueless when it comes to proper error handling" ...* I think another leading reason is that all text books and intro class so stuff like this in their examples:

```
try
{
    //sample code....
    methodCallThatThrowsSQLException();
    //sample code....
}
catch (Exception e)
{
    e.printStackTrace();
}
```

And some of them don't even put the `e.printStackTrace()`; So good exception handling techniques are not taught. I understand that a JDBC book wants to focus on JDBC code and not on "Exception handling design patterns and methodologies". But new developers get

Plugins that generate serialVersionUID...

into bad habits quickly and never end up learning how to design a good exception handling framework in their apps. Just a thought.

Good discussion. I always enjoy having discussions like this.



[constv](#) 138 posts since

Nov 8, 2002 7. Re: **Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 22, 2008 8:37 PM

Mark Vedder wrote:

constv wrote:

It's amazing how many people "program" (or should I say, insert code) without questioning why certain things are necessary and how they should be used.

How true. I once worked with a guy like this. I called him a "CGI programmer" CGI = Copies (from) Google Inquires.

Only once??? Man you are lucky! I feel blessed if I come to a project where there is one guy who is not like that! I am a self-employed consultant, and I move from project to project, and I see the same scene almost everywhere I go: "Senior Software Engineers" who can't program/design even if their lives would depend on it. It seems any idiot may suddenly decide that he can "talk Java" and be a SE. The truly good ones are 5% at most. I think that software engineering absolutely requires the combination of analytics, science, and *artistic creativity*. And artistic passion! Without all those, it is impossible to build elegant and cost-efficient solutions that just work and don't create more problems than they solve. Sadly, most SEs I see do not have a creative bone in their body and prefer a thoughtless, often dumb, approach to most problems. I love IntelliJ because it was clearly designed by creative people who are passionate about what they do. As the great Edsger Dijkstra said over and over again that *elegance in programming is not a luxury but a vital necessity*. One of my favorite quotes from Dijkstra:

"Why has elegance found so little following? That is the reality of it. Elegance has the disadvantage, if that's what it is, that hard work is needed to achieve it and a good education to appreciate it. "

As for the moronic ways of error handling - that totally prevail in Java projects all over the world, the reason is the mere fact that checked exceptions exist in the language. The concept is totally flawed. It was introduced by the people who misinterpreted the core original purpose of exceptions, which was to remove the necessity to check for error codes every step of the way and allow to consolidate error handling in a dedicated handler that has the contextual knowledge of how to handle the specific type of error. That approach would allow for all the modules between the error source and that handler to focus on the business logic instead of error handling - since they have no business of handling that error properly anyway. It would also ensure that the error info travels unaltered from the error source to the handler. Checked exceptions were invented by some opinionated people who still thought in terms of error codes that needed to be checked every step of the way in the call stack. Those people apparently - there's no other explanation - just did not understand the purpose and the idea of exceptions in a language. BTW, the exceptions mechanisms were first introduced long before Ada and C++: in early operating systems in the 60s. Gosling did not have checked exceptions in the original release of Java in 1995. They were added later because a couple of people on the board argued to death that it would be a life savior, and, unfortunately for the programming world, won the argument. Soon, it became clear to anyone who was capable of thinking for themselves - instead of just blindly following the dogma - that it was a mistake and that there was no good/correct/proper way to use checked exceptions. People spoke against them as early as in 1997, but Sun stood behind their idea, and it became an issue of pride. So, they never publicly admitted to the fact that it was a disaster. Gosling spoke in support of checked exceptions in 2003, and if you read his article (available on theserverside.com) you'll see that he really had nothing to say. His argument for the benefit of checked exceptions: they create a "creepy feeling" that something may go wrong in a module, and force the developer to do something about it. As much as I respect Gosling, that statement is horribly ridiculous! First off, *something* can *always* go wrong in *any* software module. Any developer should know that and account for! If you use the compiler to tell the developer that an IOException may occur in a module, the implication is that *nothing else* can go wrong. So, most developers will put a try/catch around that IOException and go home thinking that the job is done. But we all know (those of us who can think) that many other bad things can potentially happen (especially if the module in question is a poorly documented 3rd party module that is a black box to us.) So handling a checked exception is never enough, but that's what most people do. And that's why their apps blow up unexpectedly. Not to mention, is that, as you have pointed out, Mark, when people do catch checked exceptions, it is usually in a wrong place, and they mishandle them by throwing away the most valuable info about the cause of the error. Most Java books, no matter how good they are otherwise, contain error-handling chapters that are nothing but confusing garbage. That's because they all try to justify something for what there is no good use. Some say: use checked exceptions for recoverable errors, and RTEs for non-recoverable ones. That is such a nonsense. Who can tell what's recoverable??? How do you know what the requirements of the client are. One app may consider a certain

Plugins that generate serialVersionUID...

error recoverable, the other would want to treat it as a generic error with no alternative action. It all depends on the business cases. And if so, you as a developer must code to your business requirements and not to compiler alerts. If your requirement states that you have to implement an alternative workflow in case of an "account-already-exists" situation, then you should intentionally, thoughtfully design for that condition, and not expect to code for it only if the API throws a checked `AccountAlreadyExistsException`. For any exceptional condition that is not a legitimate use case, there should be a generic system error handler, and all such errors must *freely propagate* to it *unaltered*. That handler will treat all such errors according to the app's requirements, e.g. - in a web app - display a system error page with a friendly message, log the complete stack trace to the configured logging destination in the appropriate format, where the developers and/or admins can later pull it and investigate. Since the stack trace contains the complete error info and causes, this will be all you need. BTW, often, when you ask a person to send you an exception stack trace, send you the first few lines with a very generic message. Incredibly, even developers often don't realize that it is the very bottom of the stack trace that contains the most valuable info.

Ah well.... Sorry for the rant. 😊)) I know this discussion belongs elsewhere, but couldn't resist, once we started...

Cheers,

Constantine



[Mark Vedder](#) 556 posts since

Dec 10, 2003 8. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 23, 2008 4:53 AM

constv wrote:

Mark Vedder wrote:

constv wrote:

It's amazing how many people "program" (or should I say, insert code) without questioning why certain things are necessary and how they should be used.

How true. I once worked with a guy like this. I called him a "CGI programmer" CGI = Copies (from) Google Inquires.

Only once??? Man you are lucky! I feel blessed if I come to a project where there is one guy who is not like that!

LOL. No, trust me, it wasn't "only once"... I wish... But I've been able to use Vulcan mind meditation techniques to block out all the others. 😊 Actually I was using the "at some indefinite time in the past" [definition/usage of once](#) as opposed to "one time and no more" usage.

constv wrote:

>The truly good ones are 5% at most. I think that software engineering absolutely requires the combination of analytics, science, and artistic creativity. And artistic passion! Without all those, it is impossible to build elegant and cost-efficient solutions that just work and don't create more problems than they solve. Sadly, most SEs I see do not have a creative bone in their body and prefer a thoughtless, often dumb, approach to most problems.

I agree 100%. I'm always saying that programming is both an art and a science.

constv wrote:

>I love IntelliJ because it was clearly designed by creative people who are passionate about what they do.

I agree with you on that one.

In regards, to checked exceptions, a standard "solution" I use in (most of) my applications is simply to catch them, wrap them in a Generic "ApplicationException" that extends RTE. Then that can bubble up to a central point of control. We can use the cause if needed to make a determination. In the more complex apps we'll have some variations of the

Plugins that generate serialVersionUID...

"ApplicationException", say an ApplicationConfigurationException and a DataException. Basically similar to what Spring does. I even have an IDEA live template to make this easier.

>BTW, often, when you ask a person to send you an exception stack trace, send you the first few lines with a very generic message. Incredibly, even developers often don't realize that it is the very bottom of the stack trace that contains the most valuable info.

You mean that part that starts with the words "Caused by" is important? 😊

You've got some great points in your comments, even if we've hijacked this thread for said discussion.



[Bas Leijdekkers](#) 2,204 posts since

Aug 19, 2002 9. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Aug 25, 2008 4:21 PM

Mark Vedder wrote:

constv wrote:

My point, however, is that you don't need any special algorithm to generate such IDs. You can start with 1L, and then increment it each time you introduce updates to your class that should render the past versions obsolete. ... **1L, 2L, 3L**, etc - that is *unique for this version of the class* - compared to other versions

Ah, gotcha... I missed your comment about incrementing in your original post. Sorry about that.

As silly as it sounds, for me, I've always seen the use of a standard generator having a side benefit that I don't have to think about (i.e.

remember 😊) whether the changes I made have made the object incompatible with older versions. I just regenerate the ID. Either the same number gets generated (compatible) or a new one does. But that's probably me being lazy 😊

Plugins that generate serialVersionUID...

Note that in that case you might as well not use a serialVersionUID at all. The generator or the inspection quick fix take the entire class into consideration when generating a serialVersionUID. Any change to a method signature, even changing a 'protected' method to a 'public' method, will cause a different serialVersionUID to be created.

Usually only making changes to the fields of a class will cause compatibility problems (and even those can be worked around with the use of a readObject() and writeObject() implementation).

Bas



[Marvin](#) 1 posts since

Sep 11, 2008 10. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Sep 11, 2008 11:10 PM

With all the long posts, still nobody has answered the original question in the thread..



[Mark Vedder](#) 556 posts since

Dec 10, 2003 11. **Re: Plugins that generate serialVersionUID do not work with IDEA 8.0 M1** Sep 20, 2008 7:55 PM

Marvin wrote:

With all the long posts, still nobody has answered the original question in the thread..

My one [reply](#), in its entirety, states:

>The built in intention fix works. To use, go into your inspection settings. Under the "Serialization Issues" turn on the "Serializable class without 'serialVersionUID'" inspection. If a serializable class does not have a serialVersionUID, the class name will be highlighted with a warning. Move the cursor there and type alt+Enter to get the intention fix of "Add 'serialVersionUID' field".

>

Plugins that generate serialVersionUID...

>If you need to update the value after making changes to the class, you'll need to delete the 'serialVersionUID' field, then reuse the intention fix.

So while neither of the third party plug-ins work due to the plug-in architecture changes in IDEA 8, the built in intention fix does work.

Hope that helps.