Item 177
git_comments:

git_commits:

1. **summary:** Bug 61529 - Migration to Java 9
   message: Bug 61529 - Migration to Java 9 Add Java 9 options to avoid the warnings that will disturb our users. Warnings are due to: - https://github.com/bulenkov/Darcula/issues/41 - https://issues.apache.org/jira/browse/GROOVY-8339 Bugzilla Id: 61529 git-svn-id: https://svn.apache.org/repos/asf/jmeter/trunk@1822712 13f79535-47bb-0310-9956-ffa450edef68 Former-commit-id: 03dc1b93bbb00fc7b5e0b92ac1f21f3fd8138215

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Fix warning "An illegal reflective access operation has occurred"
   description: I'm running JDK-9 on Windows 10 with Gradle 4.2. My global gradle.properties file contains the following line: org.gradle.java.home=C:/Program Files/Java/jdk-9 When I request the gradle version (gradle --version) I get the following warning: {code:none} WARNING: An illegal reflective access operation has occurred WARNING: Illegal reflective access by org.codehaus.groovy.reflection.CachedClass (file:/C:/Program%20Files/gradle-4.2/lib/groovy-all-2.4.11.jar) to method java.lang.Object.finalize() WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.reflection.CachedClass WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations WARNING: All illegal access operations will be denied in a future release {code} This warning displayed regardless of whether I'm using a regular command prompt or an elevated rights (Administrator) command prompt. Here's the full command and output: {code:none} gradle --version WARNING: An illegal reflective access operation has occurred WARNING: Illegal reflective access by org.codehaus.groovy.reflection.CachedClass (file:/C:/Program%20Files/gradle-4.2/lib/groovy-all-2.4.11.jar) to method java.lang.Object.finalize() WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.reflection.CachedClass WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations WARNING: All illegal access operations will be denied in a future release -- Gradle 4.2 -- Build time: 2017-09-20 14:48:23 UTC Revision: 5ba503cc17748671c83ce35d7da1cffd6e24dfbd Groovy: 2.4.11 Ant: Apache Ant(TM) version 1.9.6 compiled on June 29 2015 JVM: 9 (Oracle Corporation 9+181) OS: Windows 10 10.0 amd64 {code}

jira_issues_comments:

1. **body:** That warning is to be expected on all currently released versions of Groovy under JDK9. The operation we are doing is perfectly legal under JDK8 and earlier and represents correctly working code. It just doesn't conform to the new rules imposed by the Java 9 module system. It won't affect program use under JDK9. In unreleased versions of Groovy, we have a temporary workaround to avoid the warnings involving setting an environment property flag but this doesn't apply to using Groovy in an embedded mode like from Gradle. You could adopt the same approach it uses (involves --add-opens and --add-modules) but it would involve some hackery on your part. There are other nasty hacks to get around the warning but I don't recommend them: https://stackoverflow.com/questions/46454995/how-to-hide-warning-illegal-reflective-access-in-java-9-without-jvm-argument Longer term we are looking at reworking affected parts of Groovy but it will take some time. If you don't want to hush the warnings using the --add-opens & --add-modules hack above, I'd just ignore the errors for now.
   label: code-design
2. Is there anything the users can do to assist you with resolving this issue? I ask because I would like to help, but I doubt more screenshots and reports will do anything to help at this point.
3. This issue is currently typed as an *Improvement* with a *Minor* priority. I'm concerned about the line {code} WARNING: All illegal access operations will be denied in a future release{code} and JDK 10 is scheduled to be released 20-MAR-2018. I personally doubt that JDK 10 is when the operation will begin to be denied as I don't see anything related to it in the feature list at [JDK 10 Project Page|http://openjdk.java.net/projects/jdk/10/] but I wonder if the type and/or priority should be changed. I may have been the one who assigned the type and priority initially. I believe someone closer to the project should make the determination to change the type and priority of this issue.
4. [~benrobot] the message is annoying, but that´s about it. Groovy will still work the same as on JDK9, if this no longer works in the future. That does not mean you will be able to write a module in Groovy of course.
5. This error occurs with java *10* (*18.03*) on Groovy 2.4.12 and *2.4.14* which seems to be the current Release/Production version. JDK 9 is done now. This problem appears to have effects. Plugins and tests that depend on Groovy can find they don't like the problem. ``` D:> groovy -version Groovy Version: 2.4.14 JVM: 1.8.0_162 Vendor: Oracle Corporation OS: Windows 10 D:> SET JAVA_HOME=b:\lang\java\jre\v10.00\x64 D:> groovy -version WARNING: An illegal reflective access operation has occurred WARNING: Illegal reflective access by org.codehaus.groovy.reflection.CachedClass (file:/b:/lang/groovy/v02.04/lib/groovy-2.4.14.jar) to method java.lang.Object.finalize() WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.reflection.CachedClass WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations WARNING: All illegal access operations will be denied in a future release Groovy Version: 2.4.14 JVM: 10 Vendor: "Oracle Corporation" OS: Windows 10 ```
6. [~aplatypus] Groovy continues to work fine with JDK10 despite the annoying warnings. If you notice anything not working, please let us know.
7. Does anyone know how to hide the message for now? Can I do that with `--add-opens`?
8. Take a look in startGroovy[.bat] on master. It adds the appropriate add-opens but only if an environment property is set: https://github.com/apache/groovy/blob/master/src/bin/startGroovy#L285
9. Hey guys, any idea when this will be resolved? We can't release our product with java9/10/11 because of this warning and clients are asking us about it and think there's something wrong.
10. <knock, knock, anyone home?> [INFO] >>> spotbugs-maven-plugin:3.1.9:check (findbugs) > :spotbugs @ nlp-validators >>> [INFO] [INFO] --- spotbugs-maven-plugin:3.1.9:spotbugs (spotbugs) @ nlp-validators --- WARNING: An illegal reflective access operation has occurred WARNING: Illegal reflective access by org.codehaus.groovy.vmplugin.v7.Java7$1 (file:/Users/.../.m2/repository/org/codehaus/groovy/groovy/2.5.4/groovy-2.5.4.jar) to constructor java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int) WARNING: Please consider reporting this to the maintainers of *org.codehaus.groovy.vmplugin.v7.Java7$1* WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations WARNING: All illegal access operations will be denied in a future release
11. [~ssanbeg] you can happily ignore those warnings on JDKs up until JDK12 (at least). We are working on making changes but they involve some deep re-plumbing which is taking some time.
12. [~paulk] Will do - thanks.
13. **body:** Is it worth adding some official instructions on the download page or similar? Anyone running groovy on jdk 11 will be given the idea that groovy does not work on jdk 11 - but it _sounds_ like that isn't the case? It would be useful to have the recommended workaround documented somewhere a bit

more accessible than in the comments on a Jira ticket. (this isn't just a problem for people running gradle, it's a problem for anyone running groovy in any way)
label: documentation

14. I think there should be an official instructions to explain this problem. groovy-3.0.0-alpha4 with JDK 11: {code:java} ./groovy -version WARNING: An illegal reflective access operation has occurred WARNING: Illegal reflective access by org.codehaus.groovy.reflection.CachedClass (file:/usr/local/apps/groovy-3.0.0-alpha-4/lib/groovy-3.0.0-alpha-4.jar) to method java.lang.Object.finalize() WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.reflection.CachedClass WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations WARNING: All illegal access operations will be denied in a future release Groovy Version: 3.0.0-alpha-4 JVM: 11.0.2 Vendor: Oracle Corporation OS: Mac OS X {code}
15. Yes, it is briefly mentioned in the 2.5 release notes but it would be good to provide further clarification until we fix the underlying problem.
16. I have a question regarding this issue. I started seeing this after upgrading our systems to Java 10, and Groovy 2.5.6. Our specific situation uses the map constructor to bypass private field access in Spring based applications - allowing our developers to use field injection (@Autowired) on private fields, but write unit tests that don't bootstrap the Spring application. Is there any indication at this point that the possible fixes would break this functionality?
17. The answer is we can't say just yet but such breaking functionality wouldn't be in the 2.5.x stream of releases.
18. [~elberry] [~paulk] The indications are high that as soon as we "fix" this or as soon as they turn the warning in an active prohibition your logic will no longer work unchanged. We will be required to set the fields from either within the class or with for example a Lookup object provided by the class. If the class provides no facilities, then there is no way. BUT Spring has the same problem. Thus it should be possible to use the Spring mechanism to set those. In the end I am optimistic that we find a solution for these cases
19. [~blackdrag]. That's a good point. Thanks! I'll keep an eye on this ticket for updates.
20. {{makeAccessible}} is the evil that causes most of illegal access warnings. [https://github.com/apache/groovy/blob/master/src/main/java/org/codehaus/groovy/reflection/CachedClass.java#L96] The method {{makeAccessible}} will try to call {{setAccessible}} on all declared methods, constructors and fields even if they will not be accessed actually. So I think we should postpone the invocation of {{setAccessible}} util the methods, constructors and fields are accessed. The above idea can avoid illegal access warnings unless we are truely accessing invisible objects. Here is the experiment I have tried to fix the illegal access warning of methods(Note: illegal access warning of constructors and fields can be fixed in a similar way): https://github.com/danielsun1106/groovy/commit/9720df9d616ac785756109313ab19c8983e46fcf As you can see in the following log, illegal access warning of methods(e.g. `finalize`) disappear: https://travis-ci.org/danielsun1106/groovy/jobs/516628040 Though the idea can fix illegal access warning of methods(warnings of constructors and fields can be fixed too), it also introduces a new problem, how to check the class is exported, or the following error will occur. {code:java} org.codehaus.groovy.tools.shell.GroovyshTest > testDefaultResultHookStringArray FAILED java.lang.IllegalAccessException: class org.codehaus.groovy.runtime.callsite.PlainObjectMetaMethodSite cannot access class jdk.internal.jrtfs.JrtFileSystem (in module java.base) because module java.base does not export jdk.internal.jrtfs to unnamed module @6f5cdb53 at java.base/jdk.internal.reflect.Reflection.newIllegalAccessException(Reflection.java:361) at java.base/java.lang.reflect.AccessibleObject.checkAccess(AccessibleObject.java:591) at java.base/java.lang.reflect.Method.invoke(Method.java:558) at org.codehaus.groovy.runtime.callsite.PlainObjectMetaMethodSite.doInvoke(PlainObjectMetaMethodSite.java:43) at org.codehaus.groovy.runtime.callsite.PojoMetaMethodSite$PojoCachedMethodSite.invoke(PojoMetaMethodSite.java:188) at org.codehaus.groovy.runtime.callsite.PojoMetaMethodSite.call(PojoMetaMethodSite.java:53) at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.java:47) at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:115) at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:127) at Script1.run(Script1.groovy:39) at groovy.lang.GroovyShell.evaluate(GroovyShell.java:441) at groovy.lang.GroovyShell.evaluate(GroovyShell.java:479) at groovy.lang.GroovyShell.evaluate(GroovyShell.java:450) at org.codehaus.groovy.tools.shell.util.PackageHelperImpl.getPackagesAndClassesFromJigsaw(PackageHelperImpl.groovy:149) at org.codehaus.groovy.tools.shell.util.PackageHelperImpl.getPackages(PackageHelperImpl.groovy:122) at org.codehaus.groovy.tools.shell.util.PackageHelperImpl.initializePackages(PackageHelperImpl.groovy:59) at org.codehaus.groovy.tools.shell.util.PackageHelperImpl.<init>(PackageHelperImpl.groovy:49) at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method) at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62) at java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45) at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:490) at org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:80) at org.codehaus.groovy.runtime.callsite.ConstructorSite$ConstructorSiteNoUnwrapNoCoerce.callConstructor(ConstructorSite.java:105) at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:59) at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:237) at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:249) at org.codehaus.groovy.tools.shell.Groovysh.<init>(Groovysh.groovy:108) at org.codehaus.groovy.tools.shell.Groovysh.<init>(Groovysh.groovy:97) at org.codehaus.groovy.tools.shell.Groovysh.<init>(Groovysh.groovy:125) at org.codehaus.groovy.tools.shell.Groovysh.<init>(Groovysh.groovy:129) at org.codehaus.groovy.tools.shell.Groovysh.<init>(Groovysh.groovy:133) at org.codehaus.groovy.tools.shell.GroovyshTest$1.<init>(GroovyshTest.groovy) at org.codehaus.groovy.tools.shell.GroovyshTest.createGroovysh(GroovyshTest.groovy:52) at org.codehaus.groovy.tools.shell.GroovyshTest.testDefaultResultHookStringArray(GroovyshTest.groovy:164) {code}
21. **body:** I find `trySetAccessible` can help us avoid unnecessary warnings, in addition, we need not introduce many changes to the foundation of dynamic groovy.
label: code-design
22. [~paulk] I found another illegal access warning from {{org.codehaus.groovy.reflection.InjectedInvoker}}. Do you know where is {{org.codehaus.groovy.reflection.InjectedInvoker}}? I can not find it in master branch(i.e. 3.0.0)... {code:java} WARNING: An illegal reflective access operation has occurred WARNING: Illegal reflective access by org.codehaus.groovy.reflection.InjectedInvoker/0x0000000100161840 (file:/home/travis/build/apache/groovy/target/bootstrap/groovy-3.0.0-SNAPSHOT-bootstrap.jar) to method java.util.AbstractMap.clone() WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.reflection.InjectedInvoker/0x0000000100161840 WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations WARNING: All illegal access operations will be denied in a future release {code}
23. **body:** {{InjectedInvoker}} is generated by Java... The simplified solution seems to introduce the new warnings... Setting accessible when and only when methods, constructors and fields are accessed seems a wise choice for now.
label: code-design
24. trySetAccessible is not cause the warning? Did that change?
25. **body:** [~blackdrag] {{trySetAccessible}} still causes the warnings in the generated {{InjectedInvoker}} class...
label: code-design
26. h3. *Current Status* From now on, the proposed PR([https://github.com/apache/groovy/pull/905]) can fix most of unnecessary illegal access warnings, here is the build result of Apache Groovy project: [https://travis-ci.org/apache/groovy/jobs/519678165] h3. *Cases and solutions* As you can see, there are still illegal access warnings in the build result because of the following reasons I've found: ① The code truely accesses members illegally(account for the vast majority of the warnings in the build result), e.g. [https://github.com/apache/groovy/blob/master/src/test/groovy/PrimitiveTypesTest.groovy#L88] {code:java} // groovy code BigInteger big = new BigInteger(Long.MAX_VALUE) // the constructor is private {code} *Solution:* we have to fix them by change our code to avoid illegal access ② Sub-class derives the public members from package-private class, but invoke the members on the sub class instances, e.g. [https://github.com/apache/groovy/blob/master/subprojects/groovy-groovysh/src/main/groovy/org/codehaus/groovy/tools/shell/Groovysh.groovy#L532]

{code:java} // groovy code // the declaring class of method `setLength` is `AbstractStringBuilder`(i.e. the base class of `StringBuilder`) // but `AbstractStringBuilder` is package-private buff.setLength(0) // buff is a `StringBuilder` instance {code} *Solution:* Groovy should be smarter to be able to choose the derived method, e.g. {{StringBuilder::setLength}} h3. *Help wanted* For ①, hi Groovy community, *after the proposed PR is merged*, let's add {{--illegal-access=debug}} to JVM option and find the cause of illegal access, then *fix them together*. *PRs are really welcome!* For ②, I wish [~blackdrag] could give us some hints to make progress faster ;)

27. Here is the command {{groovy -v}} and its output. Look, it is clean again, no warnings :D {code:java} C:\Users\Daniel.LAPTOP-H7RKNSIS>groovy -v Groovy Version: 3.0.0-SNAPSHOT JVM: 11.0.2 Vendor: Amazon.com Inc. OS: Windows 10 C:\Users\Daniel.LAPTOP-H7RKNSIS> {code}
28. **body:** For ②... that is not so easy... I guess when we copy the methods from the parent to the child in the meta class system for public access, we should not only consider private vs public, but also the visibility of the class itself.... in that if we "override" the method we should skip the override if the original method has better accessibility. Of course the real implementation would still be called (by callsite caching, reflection and invokedynamic) because the method call would be still virtual.
    label: code-design
29. Gotcha... As ② is not an easy task... we have to split the current Jira ticket and create a new sub-task to track.
30. Case ② now has its own issue: GROOVY-9081.
31. [~paulk] Thank you, Paul :)
32. Will this be backported to the 2.5 branch as well?
33. We'll assess that once we have confirmed our final solution. We'd typically prefer not to make such a change in our "maintenance" branch but the JVM world is moving fast and we may not have much choice.
34. All the main cases are now complete - with some special cases being handled in a cloned issue. You will still get a warning message of course if you actually do perform what the JDK now deems an illegal access. So, if you have test code for instance that is peeking into private fields or accessing private methods you will still get a warning unless you run on JDK 8 (or before).
35. **body:** Kudos to Daniel for the many hours he spent getting most of this in place.
    label: code-design
36. For the Case ②, we have made Groovy a bit smarter to find the legal path to access the members if existing. When the legal path does not exist, e.g. accessing the `protected` members of base class from sub-class, we will try to fix this kind of warnings in GROOVY-9081. But we have to admit that this kind of warnings conform to the visibility rule of Java, we just try to make Groovy "sophisticated" to avoid the warnings automatically, or groovy users have to override the `protected` members by hand, as a result, Groovy seems not that groovy then...
37. Guys also do not forget, that even though some illegal access message will then no longer appear there are still more cases than Groovy-9081. For example take a module A and have it the public API class X and one not from the public API class Y. Groovy being in the unnamed package means we can have access to X, but not Y. Now there are two basic scenarios where this is of importance. (1) To have Groovy running as module. We are far from that. (2) If Y extends X and overwrites a public method of X, then we cannot take the method from Y even if the method is public. Now you could say that we should simply go up as high as possible, but assuming it is reversed and X extends Y and X overwrites the method from Y... Then it is afaik perfectly legal to call that method on X, but not Y. In other words to really solve this, module visibilities have to be taken into account and module enabled test cases are required. The situation is actually worse considering our flow typing, in which we would tend to take the exact class. For the same reason as before, this is wrong. Y should not even been taken into consideration for a method call. Mraning the static compiler needs fixing as well in that regard.
38. [~blackdrag] Agreed, we need to start working on modules soon too.
39. Jochen, thanks for your kindly reminder :-) AFAIK, Groovy is already an automatic named module now via setting Automatic-Module-Name in MANIFEST file. For the visibility checking for modules, we have tried to complete with the following method: https://github.com/apache/groovy/blob/5d529da542befe55032e87de1f71a5010a97310b/src/main/java/org/codehaus/groovy/vmplugin/v9/Java9.java#L137 For flow type in STC, we have to discuss in the mailing list as some breaking changes may be introduced...
40. **body:** it is only an automatic module if you use it as a module (put it on the module path). The way it is used most of the time currently is as anonymous module. As for checkCanSetAccessible.... that looks very slow, but maybe it does already cover a lot of cases...
    label: code-design
41. I just remembered what was bothering and why I did not follow the route you guys currently follow. But my aim was more for allowing named Modules in Groovy... anyway: Let us assume we have the Modules M1 and M2 written in Groovy, both have public and private API. Anything in M1 is allowed to access anything in M1, anything in M2 is allowed to access anything in M2. Anything in M1 is allowed to access the public API of M2, but not the private API. Similar anything in M2 is allowed to access the public API in M1, but not the private API. Now, if you take a public API class from M1 and make a call m() to the private API of M1, this call is allowed, but it will also init the meta class for said class. Now assume you make a call from M2 to M1 and get an object back, implementing the public API, but being itself of said private API class of M1. If you now call m() you will get denied by the module system. Imagine it is not m(), but m(X) and there is also an accessible method m(Y), which would be callable as well, but X is the more precise type... Then method selection would have to use m(X) for the call from M1 and m(Y) for the call from M2. But because the meta class/ cached class is already initialized, this would not happen. Worse is the reverse case. You could think that if the init is done using what the Groovy module can see, then there is one essential point to be considered... the private APIs of M1 and M2 are not visible even to a Groovy module. That means to correctly reflect the case above not only do we have to make method selection dependent on the module, the call itself must not be done from the caller module. This rules out completely Reflection the way we use it and our cuirrent callsite caching, only leaving direct method calls and invokedynamic.
42. Understood... According to the original plan for 3.0.0, we are going to enable indy by default, but we find a performance issue(GROOVY-8298) you found some years ago, so we have to tweak the existing callsite. As the callsite will be deprecated finally in a future version, the current solution is traded off to solve most of troubles we are facing. P.S. your thoughts are always helpful to us. I wish you would go back some day, Jochen.
43. **body:** BTW, {{checkCanSetAccessible}} conforms to the rule of Java. If it is inefficient, Java is inefficient too when checking accessible ;-)
    label: code-design
44. **body:** Reflection is not known for its high speed.
    label: code-design