Base Language - Bug #7656

investigate performance of context-local helpers (e.g. TransactionManager.TransactionHelper)

08/03/2023 06:22 AM - Constantin Asofiei

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 08/03/2023 06:38 AM - Constantin Asofiei

In the profiler, I noticed that for i.e. TransactionManager$TransactionHelper.deregisterOutputParameterAssigner, there is a call to a access$... method before actually invoking TransactionManager.deregisterOutputParameterAssigner - and this access$... call takes ~230ms, from a total of ~700ms.

In a disassembler, I noticed that for this code:

```
      OutputParameterAssigner deregisterOutputParameterAssigner()
      {
         return TransactionManager.deregisterOutputParameterAssigner(wa);
      }
```

there is this bytecode:

```
     deregisterOutputParameterAssigner() { //()Lcom/goldencode/p2j/util/OutputParameterAssigner;
         <attr:org.objectweb.asm.Attribute@4853e367>

         <localVar:index=0 , name=this , desc=Lcom/goldencode/p2j/util/TransactionManager$TransactionHelper;,
sig=null, start=L1, end=L2>

         L1
             aload0 // reference to self
             getfield com/goldencode/p2j/util/TransactionManager$TransactionHelper.wa:com.goldencode.p2j.util.
TransactionManager$WorkArea
             invokestatic com/goldencode/p2j/util/TransactionManager.access$56(Lcom/goldencode/p2j/util/Transa
ctionManager$WorkArea;)Lcom/goldencode/p2j/util/OutputParameterAssigner;
             areturn
         L2
     }
```

Note the TransactionManager.access$56 method, with this definition:

```
     static synthetic access$56(com.goldencode.p2j.util.TransactionManager$WorkArea arg0) { //(Lcom/goldencode
/p2j/util/TransactionManager$WorkArea;)Lcom/goldencode/p2j/util/OutputParameterAssigner;
         L1
             aload0
             invokestatic com/goldencode/p2j/util/TransactionManager.deregisterOutputParameterAssigner(Lcom/go
ldencode/p2j/util/TransactionManager$WorkArea;)Lcom/goldencode/p2j/util/OutputParameterAssigner;
             areturn
     }
```

This is only because TransactionManager.deregisterOutputParameterAssigner is defined as private. If I change the access modifier to i.e. to package-private, then the bytecode for TransactionManager$TransactionHelper.deregisterOutputParameterAssigner is this:

```
     deregisterOutputParameterAssigner() { //()Lcom/goldencode/p2j/util/OutputParameterAssigner;
         <localVar:index=0 , name=this , desc=Lcom/goldencode/p2j/util/TransactionManager$TransactionHelper;, 
sig=null, start=L1, end=L2>

         L1
             aload0 // reference to self
             getfield com/goldencode/p2j/util/TransactionManager$TransactionHelper.wa:com.goldencode.p2j.util.
TransactionManager$WorkArea
             invokestatic com/goldencode/p2j/util/TransactionManager.deregisterOutputParameterAssigner(Lcom/go
ldencode/p2j/util/TransactionManager$WorkArea;)Lcom/goldencode/p2j/util/OutputParameterAssigner;
             areturn
         L2
     }
```

Note how know there is a direct invocation of this method.

The only reason for the synthetic access$56 method is because the TransactionManager.deregisterOutputParameterAssigner is originally defined as private.

In TransactionManager, there are some 65 access$... methods. I think it may be worth changing the access modifier from private to package-private, for methods accessed from inner classes (in TransactionManager and other parts of FWD). This is an unnecessary jump to another method call, which looks expensive on a first glance, especially if there are 100s of these methods.

I propose the following plan:

- analyze the entire FWD code, with something like this, to get all the classes with access$... synthetic methods:

```
      Class<?> cls = TransactionManager.class;

      Method[] methods = cls.getDeclaredMethods();
      for (int i = 0; i < methods.length; i++)
      {
         Method m = methods[i];
         if (m.isSynthetic() && m.getName().startsWith("access$"))
         {
            System.out.println(m.getName());
         }
      }
```

- get the list of classes with synthetic access$... methods
- go through each class with a disassembler (like Bytecode-viewer), and change the access modifier for these methods so that the synthetic methods are no longer generated by the Java compiler

#2 - 08/03/2023 08:10 AM - Constantin Asofiei

The same happens when accessing a private field: access to a private field from a non-defining class will be done via a access$... method.

#3 - 08/03/2023 08:10 AM - Greg Shah

Good find. This seems like it should have good potential.

Did you test the change for the TM to see what the performance difference is?

#4 - 08/03/2023 08:11 AM - Greg Shah

The lesson here is that compiler generated helpers are often costly.

#5 - 08/03/2023 08:13 AM - Constantin Asofiei

Greg Shah wrote:

> Did you test the change for the TM to see what the performance difference is?

Not yet. I'm working on an automated tool to analyze the FWD bytecode (but I need ASM 9 for this I think) and automatically change the field/method access modifier (maybe using spoon or just manipulating the .java file directly). It should not take long, but it will not be done today.

#6 - 08/03/2023 08:19 AM - Greg Shah

Is it worth it to manually try the changes to TM so we know the benefit?

#7 - 08/03/2023 08:23 AM - Constantin Asofiei

Greg Shah wrote:

> Is it worth it to manually try the changes to TM so we know the benefit?

Yes, I think so. There are only ~70 access$... methods, they can be manually changed. For the rest, there are ~500 classes which are reported to have access$... methods, including admin, ui, persist and other packages.

#8 - 08/03/2023 11:57 AM - Constantin Asofiei

I've tested with ControlFlowOps, ProcedureManager and TransactionManager. I don't see a definite improvement, the times are kind of spread throughout runs. I've created 7656a from trunk rev 14680, the changes are in rev 14681.

Alexandru: please apply the 7656a patch over 7156b and do a round of performance testing, maybe you'll have a more conclusive result. Thank you.

#9 - 08/04/2023 10:42 AM - Alexandru Lungu

I've done the test. The results are quite good. I tested with 7156b/rev. 14663 and your changes are close to -0.5%.

#10 - 08/04/2023 01:13 PM - Constantin Asofiei

Greg, this looks promising.

The access synthetic methods are a pitfall of using inner classes and assuming a private field/method are really visible between the inner and parent classes (or viceversa).

Otherwise, I think we need to reduce the stack depth for critical APIs (especially the cases where we go through several calls to reach the 'worker' or 'impl' method).

#11 - 08/04/2023 02:09 PM - Greg Shah

Agreed! It is a really good find.

#12 - 08/31/2023 04:51 AM - Constantin Asofiei

Alexandru, in 7656a rev 14723 I have the automated changes to modify 'private' to 'package private' fields/methods which are called between the parent and inner class (or viceversa).

The bzr diff -r 14721 patch can be applied to 7156b (there is a .rej for SchemaWorker, this can be merged manually).

Please run a performance round with these changes.

Greg: I think all fields in inner classes need to be modified to package private, not just those picked up by the script. Also, p2j.admin and p2j.ui packages are not processed at this time.

#13 - 08/31/2023 10:22 AM - Alexandru Lungu

I tested with 7156b/rev. 14732 (which was on 8.193s) patched with 7656a / rev. 14722 + 14723 and got 8.158ms. This means -0.42% improvement. This is not that different comparing to #7656-9.

#14 - 08/31/2023 10:30 AM - Constantin Asofiei

Thank you, Alexandru.

Greg: how do you want to proceed? Get the full changes or just the critical classes?

#15 - 08/31/2023 10:43 AM - Greg Shah

Full changes.

#16 - 08/31/2023 10:49 AM - Constantin Asofiei

Greg Shah wrote:

> Full changes.

OK, then I think we need this:

- for all inner classes which have fields accessed from an outer class, make them package private (so they are consistent, and not a few private and the other package-private/public).
- make it a GCD coding policy so that:

    - all inner class fields are not private. methods can be private if they are only accessed from the inner class.
    - ensure that private methods/fields defined in the outer class being accessed from inner classes - to avoid synthetic methods, they can't be private.
- include p2j.ui code in a separate branch (I want to review it first, as the changes are extensive enough in 7656a).

#17 - 08/31/2023 10:56 AM - Greg Shah

Agreed.

#18 - 11/21/2023 07:29 AM - Constantin Asofiei

Alexandru, in devsrv01:/tmp/AccessModifier.7z there is the project I use to automatically 'fix' the access modifier for fields or methods with synthetic methods.

Please apply this to the 7156b source code and run it with POC. There is a readme.txt with this content:

> This project uses the p2j/build/lib/p2j.jar to go through all classes which "access$" synthetic methods created by the javac compiler for private methods or fields accessed between an inner class and a main class.
>
> It uses spoon to automatically modify the .java source code so that the access modifier is changed from private to public.
>
> Keep in mind that the .class bytecode is used (via ASM) to determine which method/field to change, and from that, spoon is used to alter the java code. So, the reference is always the p2j.jar bytecode, and not the .java source code.
>
> To run it:
>
> 1. copy and build a p2j/ folder in the project.
> 2. java 11 JRE is required
> 3. run from within Eclipse the CheckAccessMethods2 class