Base Language - Bug #11834

make the runtime legacy class name registry context-local

09/05/2026 05:03 PM - Greg Shah

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

Description

Related issues:

Related to Base Language - Feature #3751: implement support for OO 4GL and st...	**Closed**
Related to Base Language - Feature #4373: finish core OO 4GL support	**New**

History

#1 - 09/05/2026 05:05 PM - Greg Shah

Description

ObjectOps maps a legacy class name to its converted Java class in a single JVM-wide map:

```
/** Map of legacy names converted classes. */
private static final Map<String, Class<? extends _BaseObject_>> name2cls = new HashMap<>();
```

populated with putIfAbsent:

```
private static void registerClassWorker(String cname, Class<? extends _BaseObject_> cls)
{
   ...
   synchronized (lock)
   {
      if (legacyNames.putIfAbsent(cls, cname) == null)
      {
         name2cls.putIfAbsent(cname.toLowerCase(), cls);
      }
   }
}
```

and read through the single entry point ObjectOps.resolveClass(String), whose callers include LegacyEnum, LegacyObject, Call (dynamic invoke), PropertyReference, BuilderRegistry and the test engine.

So the FIRST registration of a given legacy class name wins for the life of the JVM, for every user context in it.

Why that is wrong

A qualified class name does not identify a class: a project may declare the same one in more than one source tree and give each conversion profile the PROPATH which selects its own copy.

The 4GL keeps this per SESSION, not per process. Measured on OE 11.6:

scenario	result
session A, primary copy first on PROPATH	primary
fresh session B, secondary copy first on PROPATH	secondary

Each session resolves independently. Under FWD both sessions share name2cls, so whichever session resolves a duplicated name first binds it for every other session in that server.

Scope - what is NOT affected

Statically converted NEW emits a direct Java class reference and never consults name2cls, so ordinary converted code is unaffected. This is confirmed: converting both copies of a duplicated class produces two distinct Java classes, in packages named after their file paths, and each caller binds its own copy (see tests/conversion/duplicate_class_names/ in the testcases project).

The exposure is the dynamic-by-name paths: DYNAMIC-NEW, Progress.Lang.Class:GetClass(), dynamic invoke, CAST by name, and reflection.

Proposed change

Make the registry context-local rather than static. ObjectOps already has a ContextLocal<WorkArea>, so name2cls (and legacyNames, which is its inverse) can move into the work area, keeping the same putIfAbsent first-wins semantics WITHIN a context. That matches the measured 4GL behaviour exactly:

- first load of a name wins for that session
- a later PROPATH change does not re-resolve it
- a different session resolves independently

Care is needed for the definitions which genuinely are process-wide - the Progress built-ins, .NET and Java classes - which should stay shared rather than being duplicated into every context. The conversion side has the same split and solves it with a single shared dictionary alongside the per-PROPATH ones; see SymbolResolver.WorkArea.builtinClassDict and dictionaryFor() for the shape.

A deeper divergence, recorded but not proposed for now

The 4GL unloads a class once nothing holds it, so for a class with NO static members a later NEW after a PROPATH change re-resolves to the other copy. Measured on OE 11.6:

condition at the 2nd NEW after a PROPATH change	result
first instance still referenced	pinned to the first copy
instance deleted, no static members	re-resolves to the other copy
class has any static member (property OR method)	pinned to the first copy

This confirms the duck typing example in #3751 and explains its static constructor note: static members pin the class, so the second copy's static constructor never runs in that session.

FWD cannot reproduce the re-resolution case, because the converted Java class is fixed and there is no runtime PROPATH-driven class loading. That is a much larger change than making the registry context-local and is not proposed here; it is recorded so the limitation is known.

Testcases

Already committed in the testcases project:

- tests/oo/propath_duplicate_class/TestPropathDuplicateClass.cls - the runtime rule above, 4 tests, passing on OE 11.6
- tests/conversion/duplicate_class_names/run_duplicate_class_names.sh - the conversion side

Neither covers the cross-context bleed described here, because a single ABLUnit session cannot observe it. A test for this issue needs two contexts in one server.

#2 - 09/05/2026 05:07 PM - Greg Shah

- Related to Feature #3751: implement support for OO 4GL and structured error handling added

#3 - 09/05/2026 05:08 PM - Greg Shah

- Related to Feature #4373: finish core OO 4GL support added

#5 - 09/06/2026 02:16 PM - Constantin Asofiei

- File ducktyping.7z added

Greg, please see attached archive: duck-typing allows 2 different implementations of oo.foo, in the same session, at the same time, in different variables.

Or is this task about something else, and not duck-typing?

#6 - 09/07/2026 07:12 AM - Greg Shah

It is related to duck typing but more specifically trying to deal with things that in 4GL have duplicated fully qualified class names. The current JVM-wide mapping is a problem for such code.

Duck typing is a harder problem and would need more work. In other words, it is possible for an application (see #11746) to load the same fully qualified class name from different propaths and to isolate that code to usage in different modules/apps/areas of the code. The objects are never mixed up or swapped between these areas.

In the app in question, some of these are true duplicates (the same code, copied). In other cases, one instance is older and the other newer, with more code.

Implementing context-local for this purpose, allows this use case even if it doesn't solve the full duck-typing problem.

Would you please add the duck typing example into testcases and ensure it is used from ABLUnit?

Files

ducktyping.7z	543 Bytes	09/06/2026	Constantin Asofiei