Case Study: What’s the type of a Catch-block exception argument?

A computer code with text

Description automatically generated

Conventional employ of the one-of-three Polymorphism pillar of the OOP paradigm asks us to restrict  
the potential use-case of a specific type to a generalized type by a shared baseclass or interface.

This allows a de-coupling between the statically typed nature of C# and the ability of a use-case context to reference children of a certain hierarchy without actually knowing about all the sub-types.  
Like so, expected changes to the structure of a hierarchy do not cause changes to implementations.

However, inside of the catch-block of a try-catch statement we find the result of a “ex.GetType()” to be the output of an identical application of this method chain to the casted-from sub-type.

This adjacent situation to the conventional Polymorphism ones is caused by the retention of casted-from sub-types by reference through a bound intermediary variable inside of the catch parentheses.  
Despite an observation of the “Exception” typing of the “ex” variable, any retrieval of the type information through the implied respective method chain does not apply to the observed type at all.

This means that Polymorphism includes cases wherein a polymorphed variable does not lose any of the original typing, and merely serves an intermediary generic state including a binding to the original.  
Technically speaking, this also implies that whilst seemingly obvious, no upcast is performed here.

The typing of a catch-block exception argument is literally what’s observed as the typing of the exception argument typing, but the Runtime Type Identification (RTTI) type introspection mechanism is instructed to skip over the explicit argument typing and focus on the bound-to original instance.

In other words, we actually do refer to a polymorphed Exception variable inside of the catch-block.  
Yet, we also observe that method calls on that variable are sent down to the actual original instance.

**The answer to “what’s the output of the console in the code screenshot” would then be “the FQN of the OutOfMemoryException type” and not the seemingly plausible “FQN of the Exception type”.**

**We also learned that Polymorphism is defined by an unaware user and an origin-aware compiler;  
and not by employing either an interface or a base type. Polymorphism isn’t restricted to two forms.**