ASPECT-ORIENTED PROGRAMMING
AND EXCEPTION WRAPPING: A CASE
STUDY OF A WRAPPER LIBRARY

Yakov Yusyn
Ph. D., Assistant, Computer Systems Software Department
National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine

Tetiana Zabolotnia
Ph. D., Assistant Professor, Computer Systems Software Department
National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine

Today, aspect-oriented programming is often used as a supplement to object-oriented code - AOP practices can be found in many libraries and frameworks [1]. The main AOP practices used in conjunction with OOP are:

− cross-cutting concern – a logic that must be spread across different modules or classes (e.g., a logging capability);
− aspect – a module that encapsulates a cross-cutting concern implementation;
− join point – a place of an aspect connection to an existing code.

One of the well-known types of join points is the exception handler. In this case, the aspect's code will be called when an exception occurs in the code to which the aspect is applied. In doing so, the aspect has access to the exception object and the method's representation object (which may contain metadata about the method, any input passed, etc.). Most often, the join point of the exception handler is used for logging of information about exceptions that appear in the system, with their subsequent rethrow (Fig. 1).

However, the use of AOP with this join point can be useful in other cases, the description of such a case is the subject of this study.

Fig. 1. An example of a standard application of AOP for exception logging
In practice, sometimes there is a need to write wrapper libraries on top of other libraries. This can be caused by various reasons, for example, the need for simplified or, on the contrary, extended API of the wrapped library. A special case may be libraries that wrap libraries written in other programming languages or that wrap libraries that have been transpiled from other programming languages.

SimpleNetNlp is a wrapper library for the C# programming language that wraps the transpiled Java code of the Stanford CoreNLP natural language processing library [2]. One of the challenges when developing this library is to convert or wrap all transpiled exceptions that could appear when running the wrapped code into native exceptions of the .NET platform or custom exceptions of the library. For example, the transpiled exception `java.lang.NullPointerException` must be converted to the standard exception `System.NullReferenceException`.

The process of converting or wrapping all transpiled exceptions can be implemented with the help of various design patterns, for example, "Chain of responsibility". However, without using AOP, a call to the class responsible for this process must be added to each wrapper method, enclosing the body in a try-catch exception handling construct. As you can see, this process of converting or wrapping exceptions in AOP terms is a cross-cutting concern, and using AOP to implement this process will significantly increase the quality of the software code according to the relevant metrics.

The application of AOP for such a case is as follows:

− implementation of an aspect that, in its join point of the exception handler, performs the conversion or wrapping of the exception, followed by its throw;
− apply the implemented aspect to all wrapper methods that need to convert or wrap underlying library exceptions.

An example of the method before the application of the aspect and after its application is shown in Fig. 2.

![Fig. 2. An example of a method without and with the applied aspect](image)

At the time of implementation of AOP elements, exceptions were converted or wrapped in 7 wrapper methods (as of 31.07.2023, the number of methods using the aspect reached 34). The use of AOP affected the values of the quality metrics of the software code as follows:

− class coupling [4] – decreased from 23 to 21 (less is better);
− lines of code – decreased from 230 to 168 (less is better);
− lines of executable code – decreased from 50 to 35 (less is better);
− maintainability index [5] – increased from 80 to 87 (more is better).
Conclusions. This paper considers an atypical scenario of using the paradigm of aspect-oriented programming for converting or wrapping exceptions. The described case is relevant when writing wrapper libraries over libraries written in other programming languages or transpiled from other programming languages.

The application of aspect-oriented programming in the studied case made it possible to improve most of the standard metrics of the quality of the software code (by 8-30%, depending on the metric), even with a small number of wrapper methods.

References: