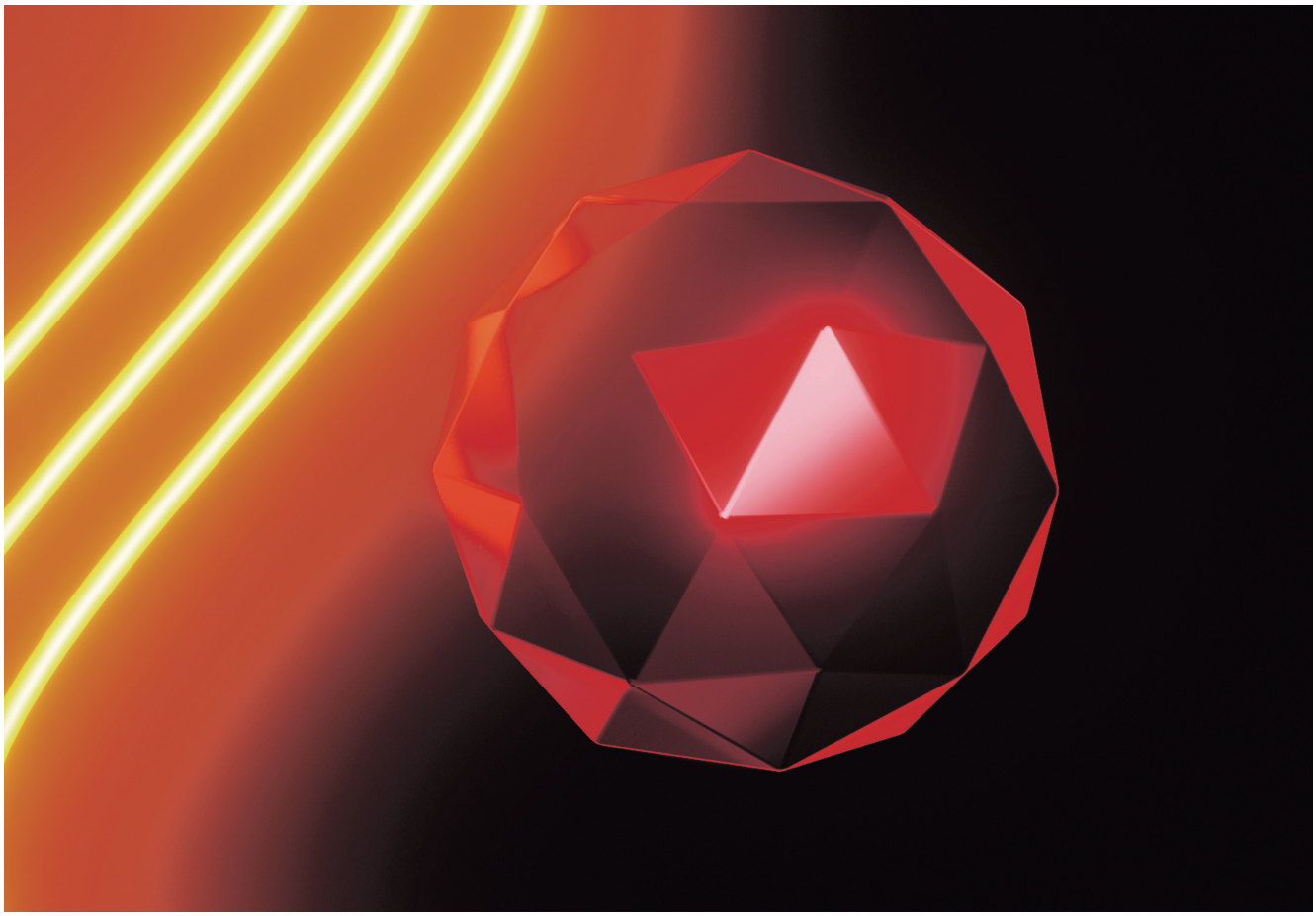


Foto: Gem

OPEN-SOURCE-PROJEKTE – LIGHT.GUARDCLAUSES

Es werde Licht!

Mit der Open-Source-Bibliothek Light.GuardClauses lassen sich Parameter einfach und robust validieren.

Im .NET-Umfeld hat sich in den letzten Jahren eine gesunde Open-Source-Community entwickelt, die durchaus interessante und sinnvolle Projekte liefert. Leider werden diese häufig aufgrund der deutlich größeren und besser vermarkteten Open-Source-Lösungen von Microsoft zu wenig beachtet.

Dieser Artikel wirft einen genauen Blick auf Light.GuardClauses [1]. Light.GuardClauses von Kenny Pflug ist eine Bibliothek, um die Werte von Parametern zu validieren. Wichtig ist hierbei, dass sowohl das Laufzeitverhalten der Funktion als auch die Lesbarkeit der Deklaration als Maß der Dinge im Mittelpunkt stehen.

Worum geht's?

Parametervalidierung ist eine der wohl undankbarsten Arbeiten beim Schreiben neuer Methoden. Trotzdem gehört die Prüfung der Eingabeparameter zu den wichtigsten Aufgaben einer Methode – stellt sie doch letztlich sicher, dass die Methode mit wohldefinierten Daten arbeitet.

Über die genaue Platzierung der Validierung lässt sich sicherlich streiten. Häufig werden nur öffentliche Methoden mit einer solchen Überprüfung ausgestattet. Weniger oft diskutiert wird die anschließend ausgeführte Aktion: Ungültige Eingabeparameter führen da nämlich zu einer Exception. Der genaue Typ des Fehlers wird hierbei durch die fehlgeschlagene Überprüfung bestimmt. Das gesamte Vorgehen ist stark von Eiffels „Design by Contract" inspiriert.

Der häufigste Fall ist sicherlich die Prüfung, ob ein Parameter überhaupt mit einem Wert belegt ist, in C# also nicht *null* ist. Klassischerweise könnte Code hier wie in Listing 1 aussehen.

Nicht nur wurden hier vier unnötige Zeilen hinzugefügt, es bleiben auch noch offene Fragen:

- Wie soll auf *null* überprüft werden?
- Was genau macht *is*?
- Welche Ausnahme soll verwendet werden?
- Wie wird der Name des Parameters übergeben? ▶

Selbstverständlich sind alle Punkte im gegebenen Schnipsel schon beantwortet worden – zumindest für den Autor. Ein anderer Entwickler hat hier möglicherweise eine ganz andere Perspektive.

Dank Light.GuardClauses könnte sich die darauf aufbauende Diskussion erübrigen. Der Code aus Listing 1 könnte damit in folgenden Code überführt werden:

```
public void Example(string name)
{
  name.MustNotBeNull(nameof(name));
  // Rest
}
```

Hier wird die Erweiterungsmethode *MustNotBeNull()* verwendet. Die Parameter für einen möglichen Ausnahmefehler können immer mitgegeben werden, sodass beispielsweise der Originalname so wie in Listing 1 über *nameof* bereitgestellt wird.

Light.GuardClauses erlaubt ein möglichst leichtgewichtiges Validieren von Eingabeparametern für den Einsatz in – zumeist internen – Methoden. Im Gegensatz hierzu gibt es auch Middleware-Bibliotheken wie FluentValidation [2], welche die komplette Geschäftslogik in Form von durchaus aufwendigen Parametervalidierungen umsetzen möchten. Die beiden Varianten sind nicht exklusiv zu betrachten.

In der Praxis

Insgesamt sind in der Light.GuardClauses-Bibliothek gut 300 verschiedene Erweiterungsmethoden enthalten. Viele Helfer gehen hier viel weiter, als nur eine knappe Überprüfung anzubieten. So wird beispielsweise C# 8 aktiv unterstützt. Dies führt dazu, dass Eingaben nicht nur auf *null* überprüft werden können, sondern anschließend auch typsicher als nicht-*null* verwendet werden können.

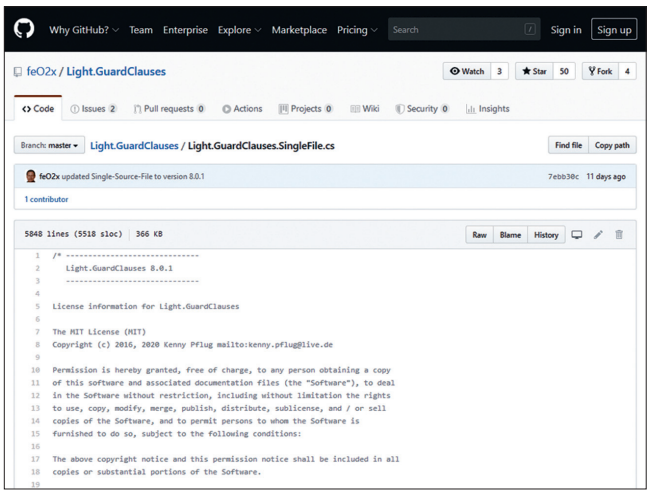
```
public class Foo
{
  private readonly IBar _bar;

  public Foo(IBar? bar)
  {
    _bar = bar.MustNotBeNull(nameof(bar));
  }
}
```

Listing 1: Klassische Parametervalidierung

```
public void Example(string name)
{
  if (name is null)
  {
    throw new ArgumentNullException(
      nameof(name));
  }

  // Rest
}
```



Die gesamte Bibliothek in einer einzigen Datei (Bild 1)

Dieses Beispiel zeigt, wie ein potenzieller *null*-Eingabeparameter einem Nicht-*null*-Feld zugewiesen werden kann. Die dafür erforderliche Arbeit erledigt die Bibliothek. Hier ist bereits alles so vorbereitet, dass der C#-Compiler alle notwendigen Angaben zur korrekten Typidentifikation vorliegen hat.

Performance hat – genauso wie sehr gute Lesbarkeit – eine hohe Priorität für die Entwicklung von Light.GuardClauses. Wer sich scheut, eine zusätzliche DLL-Datei zu referenzieren, kann die Bibliothek auch in Form einer einzigen Datei – wahlweise auch über NuGet – beziehen [3]. Bild 1 zeigt die Datei im GitHub-Repository. In diesem Fall wären sowohl die Verteilung als auch das Laufzeitverhalten und die Wartbarkeit als maximal zu betrachten.

Im folgenden Beispiel ist der Einsatz von Light.GuardClauses jenseits von einfachen *null*-Überprüfungen zu sehen:

```
public void SetMovieRating(Guid movieId,
    int numberOfStars)
{
  movieId.MustNotBeEmpty();
  numberOfStars.MustBeIn(Range.FromInclusive(0).
    ToInclusive(5));
  // Rest
}
```

Hier wird das volle Spektrum der Bibliothek ausgenutzt. Zwar erreicht man hiermit, wie bereits angedeutet, nicht den Validierungsgrad von Middleware-Bibliotheken. Dies ist aber auch weder gewünscht noch sinnvoll.

Einen Beitrag leisten

Bereiche für Community-Beiträge zum Projekt gibt es viele. Neben dem Dauerbrenner-Thema Dokumentation sind in

„Ich will mich auf de Funktionalität konzentrieren“

Kenny Pflug, der Autor der Bibliothek Light.GuardClauses, erläutert im Interview, warum er die Bibliothek geschrieben hat und wie es in Zukunft weitergehen könnte.

Kenny, wie sieht dein Hintergrund aus?
Kenny Pflug: Ich bin Senior Software Developer bei der Synnotech AG in Regensburg. In meiner Freizeit arbeite ich an einer Doktorarbeit für die Universität Regensburg, bei der ich die Deserialisierung von komplexen Objektgraphen untersuche, ohne dass man dabei Data Transfer Objects benötigt. Ich spiele gerne Gitarre und Videospiele. Seit 2016 entwickle ich an der Open-Source-Bibliothek Light.GuardClauses.

Warum hast du Light.GuardClauses erstellt?
KP: Wie jeder Entwickler habe auch ich früh gelernt, dass das Validieren von Parametern sinnvoll ist. Aber wenn ich eine Methode schreibe, dann will ich mich auf die eigentliche Funktionalität im Methodenkörper konzentrieren. Das Parameter-Validieren lenkt mich davon ab: Schon wieder ein Not-Null-Check? Welche Exception sollte ich hier schmeißen? Wie formuliere ich die Fehlermeldung so, dass sie den Aufrufer eindeutig und schnell auf den Fehler hinweist? Deswegen habe ich mir in früheren Projekten Methoden angelegt, die das Überprüfen von Parametern und das Schmeißen einer Exception mit sinnvoller Fehlermeldung für mich übernehmen. Und irgendwann habe ich das Ganze als Open-Source-Projekt umgesetzt und auf NuGet bereitgestellt.

Für welche anderen Open-Source-Projekte leistest du noch Beiträge?
KP: Ich muss gestehen, dass ich hauptsächlich für meine eigenen Open-Source-Projekte Code schreibe. Das meiste, was ich zu anderen Projekten beisteuere, sind Bug Reports, Feature-Vorschläge oder Fragen via Issues. Ab und zu liefere ich kleine Beiträge, die aber meistens nur sehr kleine Pull Requests werden. Hoffentlich wird dies besser, wenn ich die Doktorarbeit abgeschlossen habe.

Was ist deine Vision für die Zukunft von Light.GuardClauses?
KP: Die neue Version 8 vom März 2020 brachte Unterstützung für C#-8-Nullable-Reference-Typen. Für zukünftige Versionen könnte ich mir vorstellen, dass die Fehlermeldungen lokalisiert werden können. Ebenso denke ich darüber nach, die Validierung von Data Transfer Objects zu unterstützen, ganz konkret für IValidatableObject.

Welche Features fehlen aus deiner Sicht noch in C# und .NET?
KP: Ich würde mich sehr freuen, wenn Shapes Einzug in C# halten. Dies sind Abstraktionen, die kein Objekt benötigen, wie das aktuell bei Interfaces oder Delegates der Fall ist. Man kann dabei vorschreiben, dass bestimmte Typen eine statische Methode enthalten. Das hilft, um zum Beispiel eine allgemeine Abstraktion über alle numerischen Datentypen zu schreiben, was High-Performance-Code ohne viel Codeduplizierung ermöglicht.

Vielen Dank für das Interview!

diesem Fall vielfach auch Öffentlichkeitsarbeit und Tutorials denkbar.

Technisch gesehen gibt es hier einiges mit Analyzern für Roslyn und externen Tools (beispielsweise FxCop oder ReSharper) zu tun. Sicherlich spielen durch die Unterstützung von C# 8 manche Themen keine so große Rolle mehr wie zuvor. Dem Anspruch oder der Sinnhaftigkeit tut dies jedoch keinen Abbruch.

Als Dokumentationsbasis wird das im GitHub-Projekt enthaltene Wiki genutzt. Hier steht auch eine Seite mit einer ausführlichen Anleitung hinsichtlich der Weiterentwicklung zur Verfügung. Neben Ideen, Tipps und Anregungen für entsprechende Tests von neuen „guard clauses“ finden sich hier die Branching-Philosophie und eine kurze Skizze des Repository-Inhalts.

Einschätzung

Jeder Entwickler muss Parameter sinnvoll und sorgfältig validieren. Dank der ausgetüftelten Verteilungsmethoden von Light.GuardClauses gibt es keinerlei Grund mehr, unzureichende Validierungsmechanismen zu verwenden. Einfachheit war bei der Entwicklung der Bibliothek eine Motivation von Autor Kenny Pflug, siehe auch das Interview mit ihm. Im Single-File-Modus ist die Bibliothek äußerst schnell installiert, aktualisiert und im Projekt einsetzbar.

Wer eine aufwendige Validierung für sein Web API sucht, ist bei Projekten wie FluentValidation besser aufgehoben. Beide Projekte funktionieren jedoch wegen der doch unterschiedlichen Einsatzbereiche prächtig nebeneinander. Während FluentValidation die Validierung von Parametern zwischen Kunde und Code übernimmt, lässt sich Light.GuardClauses zur Validierung der Parameter zwischen Entwickler und Code sehr leichtgewichtig und zuverlässig einsetzen. ■

[1] Light.GuardClauses, www.dotnetpro.de/SL2012HiddenGems1
[2] FluentValidation, https://fluentvalidation.net
[3] Light.GuardClauses.SingleFile.cs, www.dotnetpro.de/SL2012HiddenGems2

Dr. Florian Rappl
ist Solution Architect bei smapiot mit Einsatzgebiet digitale Transformation. Als Microsoft MVP für Entwicklungstools befasst er sich mit Themen wie C#/.NET, TypeScript sowie skalierbaren Backends und Frontends in der Cloud.
@FlorianRappl

dnpCode A2012HiddenGems