Wie viele kleine Tricks und Tastenkürzel meine tägliche Arbeit am Code effektiver machen

von Martin Lippert

Your Daily Dose of QuickFix and Refactoring

Während viele ihre Eclipse-Installation durch eine ganze Anzahl von zusätzlichen Plug-ins aufpeppen, entdecke ich jeden Tag neue Tipps und Tricks, die schon die Standard-Installation von Eclipse mitbringt und die meine tägliche Arbeit mit Java-Code um so vieles effizienter gestaltet. Das Java-Tooling in Eclipse enthält eine Unmenge an nützlichen Tastenkürzeln und kleinen Features. Richtig eingesetzt können sie die eigene Entwicklungsgeschwindigkeit um ein Vielfaches steigern. Vor allem die so genannten QuickFixes und die kleinen Refactorings in Eclipse haben meine Art und Weise, Code zu schreiben, grundlegend verändert. Wie? Das wollen wir uns im Folgenden näher betrachten.

Um meine Lieblings-QuickFixes und -Refactorings zu demonstrieren, beschreibe ich im Folgenden, wie ich eine kleine Klasse testgetrieben entwickle. (Der Fokus des Artikels liegt jedoch nicht auf der Vorgehensweise testgetriebener Entwicklung. Aus der TDD-Perspektive würde man sicherlich einige der folgenden Entwicklungsschritte in einer anderen Reihenfolge durchführen.) Um die vielen kleinen Tipps und Tricks nachzuvollziehen, bietet es sich an, sich mit diesem Artikel zur Seite vor eine laufende Eclipse-IDE zu setzen und das Bespiel direkt beim Lesen „mitzumachen". Also: Ran an die Tastatur und auf zum virtuellen Pair Programming.

Porträt

Martin Lippert ist langjähriger Consultant und Coach in den Bereichen agile Methoden, Softwarearchitekturen und Java. Er berät Entwicklungsorganisationen bei der Einführung und Anwendung agiler Entwicklungsmethoden und hilft Teams, neue Wege in der Softwareentwicklung zu bestreiten. Neben großen und komplexen Refactoring gehört die Entwicklung großer plug-in-basierter Anwendungssysteme auf Basis der Eclipse-Plattform zu seinen Spezialgebieten. E-Mail: lippert@acm.org, www.martinlippert.com

QuickFix: Create Class

Beginnen wir mit einer noch leeren Test-Klasse, in der eine noch leere Test-Methode steht. Diese Methode soll nun die Funktionalität einer (noch nicht existenten) Klasse *Film* testen. Dazu erzeuge ich mir zunächst ein entsprechendes Objekt, indem ich *new Film()* schreibe (ohne eine lokale Variable oder eine Zuweisung einzutippen). Natürlich habe ich die Klasse *Film* noch nicht angelegt. Sie existiert noch gar nicht. Ich habe zunächst nur die Objekterzeugung hingeschrieben. Natürlich merkt Eclipse an, dass es den Typ *Film* noch nicht kennt. Jetzt drücke ich Ctrl + 1 (die gängige Abkürzung für QuickFix) und wähle Create class 'Film' aus. Nachdem der Wizard für eine neue Klasse erscheint, kann ich durch einen einfachen Druck auf Enter die Klasse erzeugen.

QuickFix: Assign Statement to New Local Variable

Nachdem mir der Wizard eine neue Klasse generiert hat, gehe ich mit Alt + Links zurück auf die Ausgangsklasse. Da ich das gerade neu erzeugte Objekt der Klasse verwenden möchte, muss eine lokale Variable her, in der ich mir eine Referenz auf das *Film*-Objekt merke. Dazu gehe ich mit dem Cursor in die Zeile, die das *new Film()* enthält und drücke wieder Ctrl + 1. Diesmal bietet mir Eclipse (neben einigen anderen Möglichkeiten) an: Assign statement to new local variable. Das veranlasst Eclipse dazu, die passende Deklaration einer lokalen Variable inklusive deren Zuweisung in den Code einzufügen. Dabei macht Eclipse sowohl einen Vorschlag für den Typ der Variablen als auch für deren Namen. Anhand der blauen Umrandung um beide Vorschläge erkenne ich, dass ich mittels Tab zwischen beiden Stellen springen und, wenn ich möchte, einen anderen Typ oder einen anderen Bezeichner vergeben kann. In der Regel sind die Vorschläge von Eclipse aber recht gut. In einigen Fällen bietet Eclipse auch Alternativen an, die dann durch eine Pop-up-Box angezeigt werden, wenn man mit Tab an die entsprechende Stelle springt. Um diesen Editier-Modus mit den blauen Umrandungen zu verlassen, drücke ich meist Escape.

QuickFix: Create Method

Im nächsten Schritt möchte ich an meinem *Film*-Objekt die (noch nicht existente) Methode *getTitel()* aufrufen. Dazu schreibe ich in die nächste Zeile einfach den Methodenaufruf in meinen Code: *film.getTitel()*. Wieder erinnert mich Eclipse daran, dass diese Methode an der Klasse *Film* noch nicht existiert. Durch einen gezielten Druck auf Ctrl + 1 wähle ich den QuickFix Create method ´getTitel()´ in Film .java aus. Eclipse springt daraufhin in die Klasse *Film* und generiert eine leere Methode *getTitel()*. Das soll uns zunächst reichen. Wir springen zurück in die Testklasse (Alt-Links).

QuickFix: Create Constructor

Als Nächstes wollen wir den Film gleich mit einem Titel erzeugen. Dazu ergänzen wir den Aufruf des Konstruktors in der Testklasse einfach um einen passenden Titel: *new Film("MyMovie")*. Natürlich bemerkt Eclipse, dass es einen solchen Konstruktor in der Klasse *Film* noch nicht gibt. Deshalb wählen wir nach einem Druck auf Ctrl + 1 den Quick-Fix Create constructor ´Film (String)´.

Eclipse sprint daraufhin in die Klasse *Film* und generiert einen Konstruktur mit einem String-Parameter. Der Name für den Parameter hat Eclipse mit *string* vorbelegt, was uns noch nicht gefällt. Also zweimal auf Tab drücken und einen sprechenderen Namen eingeben, wie beispielsweise *titel*.

QuickFix: Assign Parameter to New Field

Die Implementierung des Konstruktors ist noch leer. Allerdings wollen wir den Parameter gleich auf eine neue Exemplarvariable der Klasse *Film* zuweisen. Um nicht alles dafür eintippen zu müssen, hilft wieder ein Druck auf Ctrl + 1, während wir mit dem Cursor noch auf dem Parameter *titel* stehen. Eclipse bietet uns an, eine neue Exemplarvariable zu erzeugen und eine passende Zuweisung in den Konstruktor zu generieren.

QuickFix: Add Return Statement

Um die Klasse *Film* vorerst zu komplettieren, verändern wir die Methode *getTitel* noch so, dass sie einen String als Rückgabewert liefert. Eclipse bemerkt ordnungsgemäß, dass in der Methodenimplementierung ein Return Statement fehlt. Durch Ctrl + 1 können wir uns auch dies generieren lassen. Eclipse errät wiederum, den Wert welcher Exemplarvariable man wohl zurückgeben möchte. In der Regel erscheint eine Pop-up-Box, in der man aus Alternativen auswählen kann.

Refactoring: Extract Local Variable

Kehren wir zurück zur Testklasse und klammern den Aufruf von *getTitel* in eine *assert*-Anweisung ein. Wir prüfen damit den Rückgabewert auf den Wert, den wir im Konstruktor dem Film mitgegeben haben. Wir gelangen zu dem Aufruf: *assertEquals("MyMovie", film.getTitel)*.

Da wir mit dem Titel des Films noch weitere Aktionen vorhaben, wollen wir uns in der Testmethode den Titel des Films in einer lokalen Variablen merken. Dazu selektieren wir den kompletten Ausdruck *film.getTitel()* und drücken Alt + Shift + L. Dadurch rufen wir das Refactoring Extract Local Variable auf. In dem folgenden Dialog können wir den Namen der neuen lokalen Variable eingeben. Wenn der gleiche Ausdruck mehrfach in der Methode vorkommt, ersetzt Eclipse auf Wunsch alle Vorkommnisse durch die neue lokale Variable.

Refactoring: Convert Local Variable to Field

Wir stellen fest, dass die bisher noch lokale Variable *film* besser eine Exemplarvariable der Testklasse werden soll. Dazu gehen wir mit dem Cursor auf die Variable und drücken Alt + Shift + F. Eclipse bietet uns einen Dialog an, mit dem wir die lokale Variable in eine Exemplarvariable umwandeln können. Für den Namen der Variable schlägt Eclipse den Namen der lokalen Variablen vor.

Weitere QuickFixes und Refactorings

Ähnlich wie die bisher dargestellten QuickFixes bietet Eclipse zu vielen Problemen und Fehlern entsprechende Fixes vor. Bei einer Problemmarkierung in Eclipse lohnt sich also immer ein Druck aus Ctrl + 1, um mal nachzusehen, was Eclipse anbietet. Neben den hier dargestellten QuickFixes gehört natürlich Rename in file zu meinem absoluten Lieblings-QuickFixes. Es erlaubt, einen Bezeichner in dem aktuellen File umzubenennen. Dabei werden alle Stellen, an denen dieser Bezeichner vorkommt, synchron mit verändert, während man den neuen Namen des Bezeichners eingibt. Allerdings handelt es sich hierbei nicht um ein echtes Refactoring. Es werden nur die Verwendungen des Bezeichners verändert, die in der gleichen Datei vorkommen. Referenzen auf diesen Bezeichner außerhalb der aktuellen Datei werden nicht aktualisiert. Daher eignet sich dieser QuickFix hauptsächlich für lokale oder private Exemplarvariablen. Klassen würde ich immer über das Rename-Refactoring (Alt + Shift + R) von Eclipse umbenennen. Das Refactoring-Feature sichert mir zu, dass alle Referenzen auf diese Klasse im gesamten Workspace aktualisiert werden und die Änderung am Code somit das Verhalten meines Systems nicht verändert. Gleiches gilt auch für Methodennamen, die sich mit dem Rename-Refactoring spielend (auch in Klassenhierarchien) umbenennen lassen.

Neben dem Rename-Refactoring gehört Extract Method zu dem absoluten Standard-Refactoring-Repertoire der täglichen Arbeit. Nichts macht es mir so einfach, eine komplexe und/oder zu lange Methode in kleinere Methoden aufzuteilen.

Fazit

Früher war ich gewohnt, lokale Variablen per Hand zu definieren oder Zuweisungen immer explizit hinzuschreiben, inklusive Typ und Namen des Bezeichners. Heute schreibe ich beispielsweise fast keine Zuweisung auf eine lokale Variable mehr per Hand. Ich rufe die Methode und lasse mir das Ergebnis automatisch auf eine lokale Variable zuweisen. Ähnliches gilt auch für die anderen QuickFixes und Refactorings, die ich hier kurz skizziert habe. Meine Arbeit mit Code hat sich grundlegend verändert und – vor allem – beschleunigt. Und ich entdecke auch heute noch immer wieder neue QuickFixes und Tastaturabkürzungen, die die Arbeit am Code so einfach machen. Ich bin schon gespannt auf die nächsten QuickFixes und Refactoring-Features, die mir begegnen werden. Sie auch?