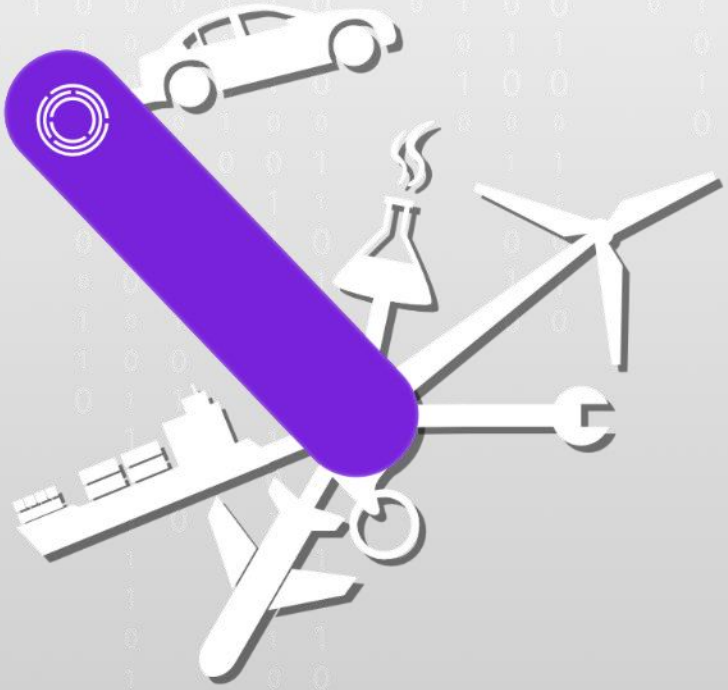
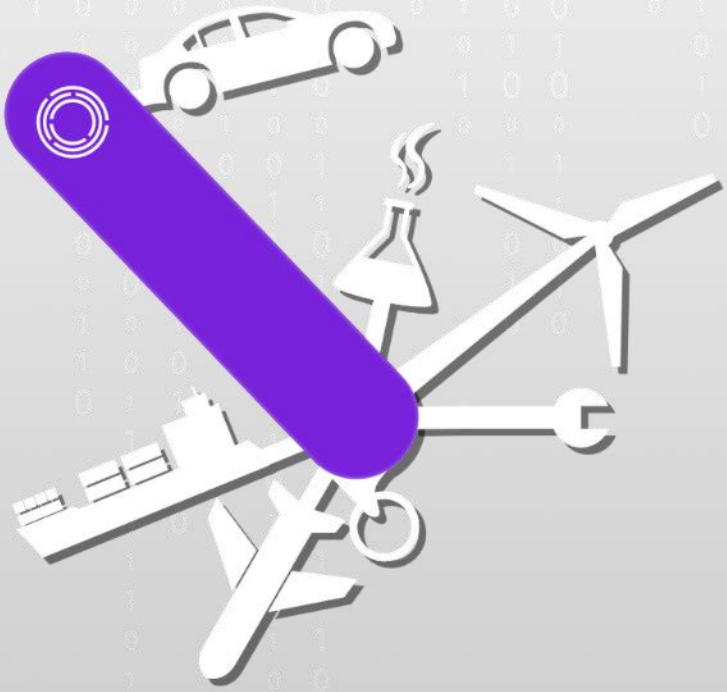


OPTANO
a KEARNEY company

Paderborn 03|09|2024

# Refactoring

Dr. Stephanie Behrent



OPTANO
a KEARNEY company

# Agenda

## Was ist das?

Definition

## Was soll das?

Gründe, Vorteile

## Wann mach ich das?

Zeitpunkte für Refactoring

## Wie geht das?

Techniken

## Workshop

Refactoring an einem Code-Beispiel

# Was ist das?

OPTANO
a KEARNEY company

## Definition

- Prozess zum Verbessern der
  - Performance
  - Lesbarkeit
  - Komplexität
  - Wartbarkeit
  - Testbarkeit des Codes
- indem man die interne Struktur verändert und die (Kern-) Funktionionalität unverändert lässt

# Was soll das?

OPTANO
a KEARNEY company

## Gründe für Refactoring

- Codequalität sicherstellen
- Testbarkeit erhöhen
- Codeverständnis
- Identifizierbarkeit von Fehler
- Performance (Probleme leichter zu identifizieren und zu beheben)
- Erweiterbarkeit vereinfachen
- Code-Smells reduzieren

# Wann mach ich das?

OPTANO
a KEARNEY company

## Zeitpunkte

- mit der Umsetzung/vor Abschluss eines Tasks
- als fester Bestandteil im TDD (Red, Green, Refactor)
- beim Schreiben von Tests
- beim Hinzufügen eines neuen Features
- beim Arbeiten mit Legacy-Code

# Wie geht das?

OPTANO
a KEARNEY company

## Methoden/Werkzeuge

- bestehende "Rezepte" nutzen
- IDE Unterstützung nutzen
- Sicherheitsnetz aus Tests

# Rezepte

OPTANO
a KEARNEY company

## Rename

Umbenennung von Variablen, Methoden, Klassen, Parametern,...

- Erhöht Verständnis
- Verbesserte Lesbarkeit
- Aufgabe des Codes klar erkennbar

Refactor → Rename (STRG + R, R)

# Rezepte

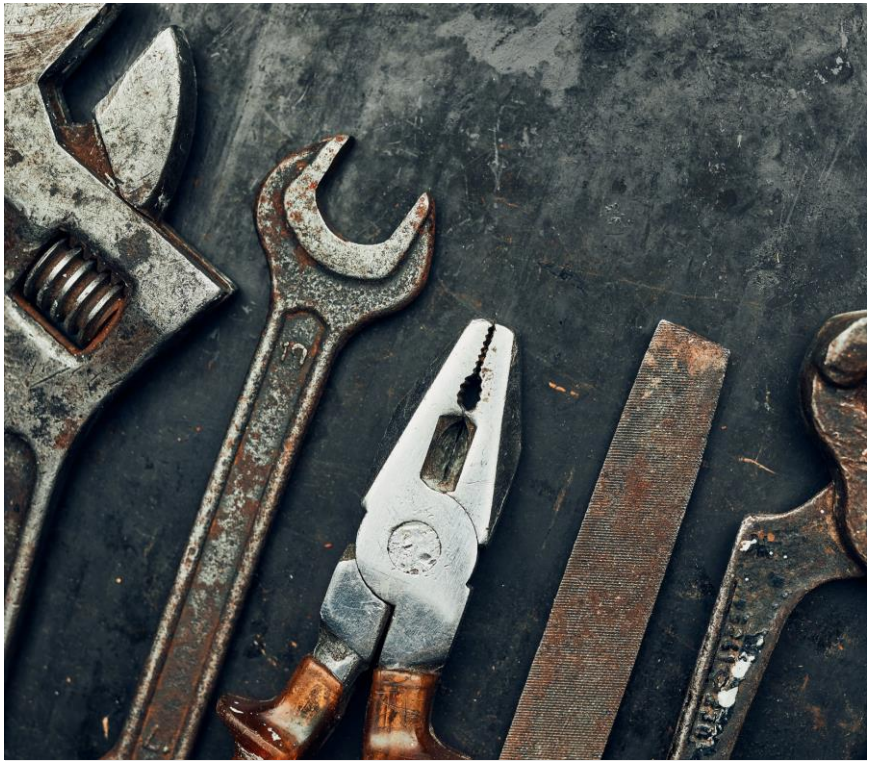
OPTANO
a KEARNEY company

## Extract Method

lange Methoden in kleinere Methoden runterbrechen

- wiederverwendbar
- Vermeidung von Duplikaten
- eine klare Aufgabe
- testbar
- gute Benennung möglich

Refactor → Extract Method (STRG + R, M)



# Rezepte

OPTANO
a KEARNEY company

## Extract Interface

Aufgabe/Sichtbarkeit extrahieren, Schnittstellen zu anderem Code festlegen

- mehrere unterschiedliche Implementierungen möglich
- Testbarkeit erhöht (Mock erstellen)
- Unabhängigkeit von Code (Decoupling)

Refactor → Extract Interface (STRG + R, I)

# Rezepte

OPTANO
a KEARNEY company

## Extract (Super-)Class

Extrahieren von Aufgaben in eine eigene Klasse

- Single Responsibility sicherstellen
- Testbarkeit erhöht
- Wiederverwendbarkeit ermöglichen

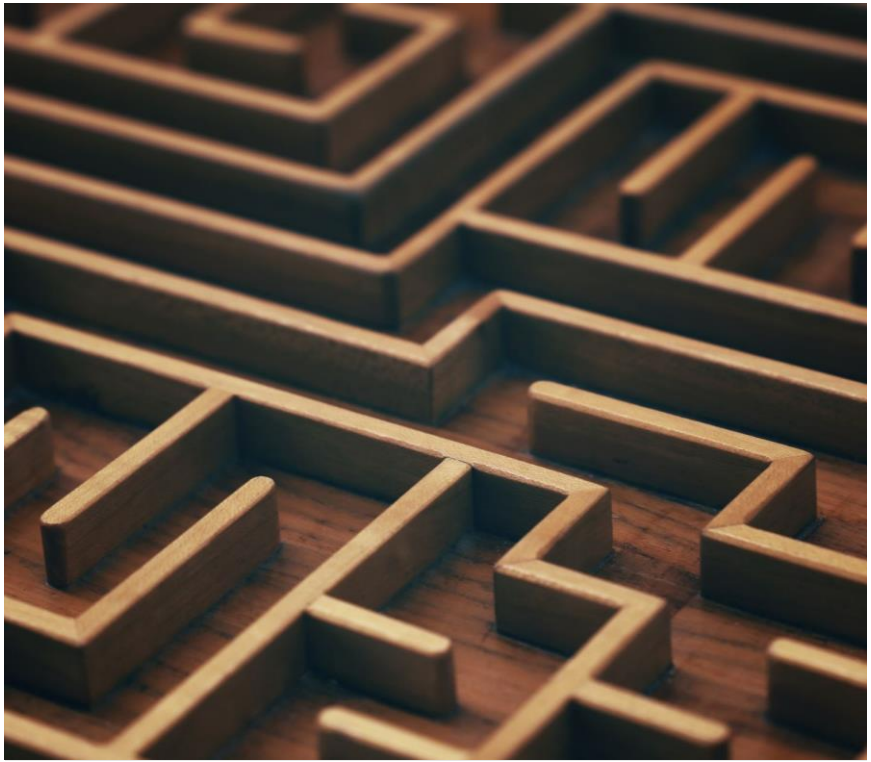
# Rezepte

OPTANO
a KEARNEY company

## Replace Conditional with Polymorphism

komplizierte/verschachtelte Bedingungsabfragen ersetzen durch Klassen (mit entsprechendem Interface), die die Logik kapseln

- lesbarer
- Aufgabe und Erwartung der einzelnen Klassen klar
- Logik kann an mehreren Stellen verwendet warden
- Erweiterbarkeit gegeben



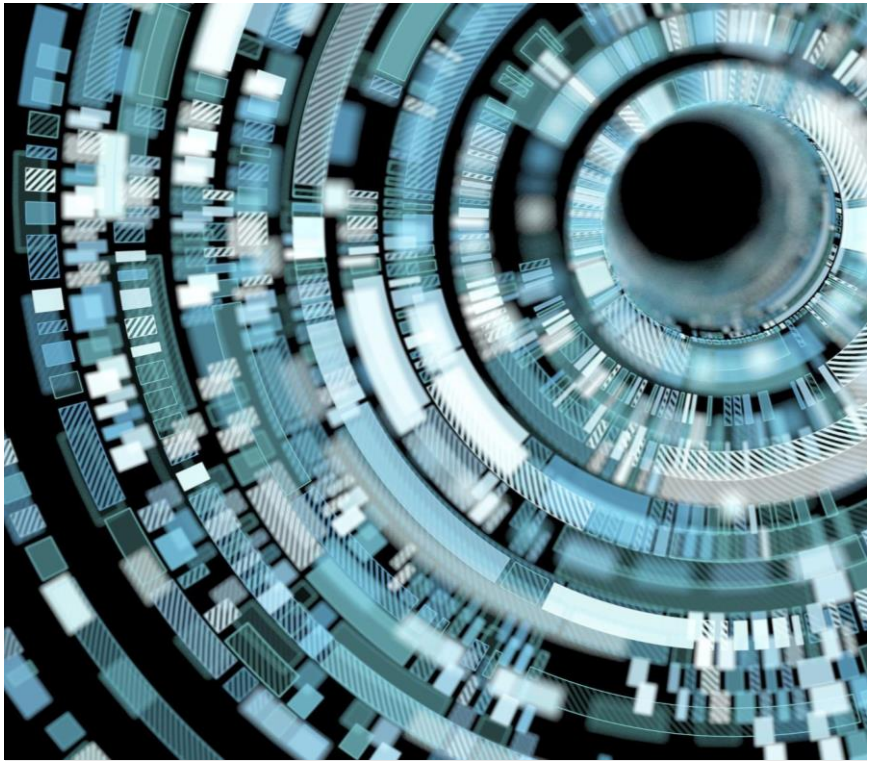
# Wie geht das?

OPTANO
a KEARNEY company

## Code-Beispiel

Refactoring am Code-Beispiel

https://github.com/emilybache/GildedRose-Refactoring-Kata/blob/main/csharp.NUnit/GildedRose



# Anforderungsspezifikation

Hallo und willkommen im Team Gilded Rose. Wie Du sicher weißt, sind wir ein kleiner Gasthof in bester Lage in einer bekannten Stadt, der von einem freundlichen Gastwirt namens Allison geführt wird. Wir kaufen und verkaufen nur die besten Produkte. Leider verschlechtert sich die Qualität unserer Waren ständig, da sie sich ihrem Mindesthaltbarkeitsdatum nähern. Wir haben ein System eingerichtet, um den Bestand automatisch aktualisieren zu können. Es wurde von Leeroy entwickelt, ein vernünftiger Typ, der zu neuen Abenteuern aufgebrochen ist. Damit wir mit dem Verkauf eines neuen Produkttyps beginnen können, ist es nun Deine Aufgabe, unserem System eine neue Funktion hinzuzufügen.

Zunächst eine Einführung in unser bestehendes System:

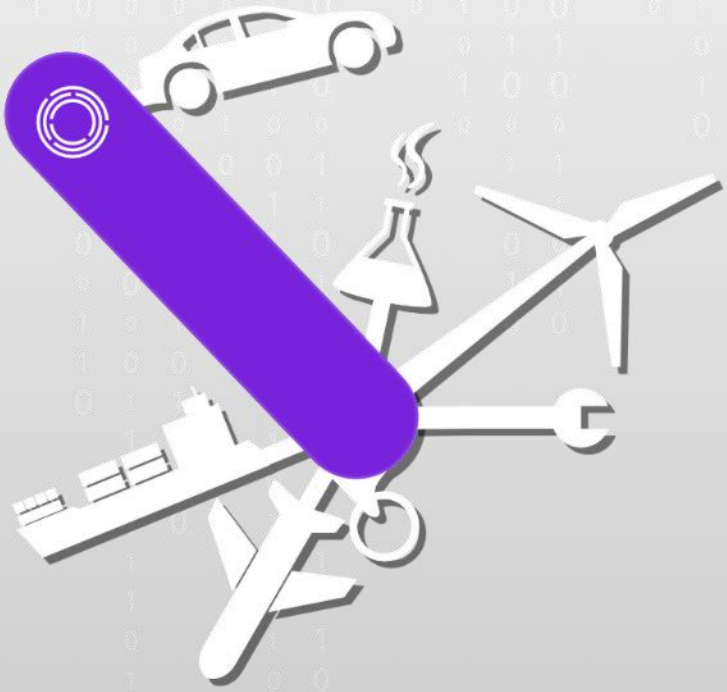
- Alle Artikel (Item) haben einen SellIn-Wert, der die Anzahl der Tage angibt, die uns verbleiben, um den Artikel zu verkaufen
- Alle Artikel haben einen Quality-Wert (Qualität), der angibt, wie wertvoll der Artikel ist
- Am Tagesende senkt unser System für jeden Artikel beide Werte

Ziemlich einfach, oder? Nicht ganz, denn jetzt wird es interessant:

- Sobald das "Mindesthaltbarkeitsdatum" überschritten wurde, nimmt die „Qualität“ doppelt so schnell ab
- Die "Qualität" eines Artikels ist nie negativ
- "Alter Brie" (*Aged Brie*) nimmt an Qualität zu, je älter er wird
- Die "Qualität" eines Artikels ist nie höher als 50
- Der legendäre Artikel "*Sulfuras*" ändert weder sein "Verkaufsdatum", noch verschlechtert sich seine "Qualität"
- "Backstage-Pässe" (*backstage passes*) werden - wie Aged Brie - hochwertiger, solange das "Verkaufsdatum" noch nicht erreicht wurde. Bei 10 Tagen oder weniger erhöht sich die Qualität um 2, bei 5 Tagen oder weniger um 3, nach dem "Konzert" sinkt sie aber auf 0.
- Kürzlich haben wir einen Lieferanten für "beschworene" (*conjured*) Artikel unter Vertrag genommen. Dies erfordert ein Update unseres Systems: "Beschworene" Artikel verlieren doppelt so schnell an Qualität wie normale Artikel

Solange alles einwandfrei funktioniert, kannst Du beliebige Änderungen an der Methode updateQuality vornehmen und so viel Code hinzufügen, wie Du möchtest. Aber Vorsicht: Die Item-Klasse oder ihre Eigenschaften darfst Du in **keiner** Weise ändern, denn diese Klasse gehört dem Kobold in der Ecke, der sofort wütend wird und Dich sofort töten würde, denn er glaubt nicht an die Kultur von gemeinsamem Code (shared code). (Wenn Du möchtest, kannst Du die updateQuality-Methode und die Item-Eigenschaft statisch machen, das regeln wir dann.)

Sicherheitshalber noch ein Hinweis: Die Qualität eines Artikels kann nie höher als 50 sein, aber Sulfuras ist ein legendärer Artikel und als solcher beträgt seine Qualität 80 und ändert sich auch nie.



OPTANO
a KEARNEY company

## Copyright