Workshop 01

Workshop 01

Dies ist eine kurze Schritt für Schritt Einleitung

To Do 1: Extract Object Method

Um eine innere Klasse für die lange Funktion zu erzeugen

testableHtml Methode Body markieren

```
  WikiPage wikiPage = pageData.getWikiPage();
  ... // bis
  return pageData.getHtml();
```

- Rechter Maustaste => Refactoring => Extract Object Method
- Name = TestableHtmlMaker
- Parameters: PageData pageData, boolean includeSuiteSetup

To Do 2: Extract Fields...

Um aus lokalen Variablen Instanzvariablen machen. Dies sind die beiden lokalen Variablen in der *invoke()* Methode

```
public String invoke() {
        WikiPage wikiPage = pageData.getWikiPage();
        StringBuilder buffer = new StringBuilder();
```

- Variable *wikiPage* markieren
- Rechter Maustaste => Refactoring => Extract => Field..
- *wikiPage = pageData.getWikiPage()* in den Konstruktur verschieben
- Variable *StringBuilder buffer* markieren
- Rechter Maustaste => Refactoring => Extract => Field..
- *buffer = new StringBuilder()* in den Konstruktur verschieben.

To Do 3: Repeatin Structure Eliminieren

Mit Extract Field... & Extract Variable ... können wir möglicherweise aus einem Methodenaufruf eine Instanzvariable extrahieren und ggf. Aus einem Konstanten eine lokale Variable extrahieren, um Repeating Structure zu eliminieren.

Die Idee ist, Repeating Structure auf einen gemeinsamen Nenner zu bringen, um daraus eine parameterizierte Methode zu extrahieren.

Folgende Struktur kommt 4 mal in ähnlichen Varianten vor:

```
WikiPagePath pagePath =
     wikiPage.getPageCrawler().getFullPath(suiteSetup);

String pagePathName = PathParser.render(pagePath);

buffer.append("!include -setup . ")
.append(pagePathName).append("\n");
```

- *wikiPage.getPageCrawler()* markieren
- Rechter Maustaste => Refactoring => Extract => Field..
- Name = *crawler* und für das 3x Vorkommen übernehmen
- *crawler* = wikiPage.getPageCrawler() in den Konstruktur verschieben.

Jetzt aus Konstanten lokalen Variablen machen (Parameterization)

"setup" String markieren

```
buffer.append("!include -setup . ")
.append(pagePathName).append("\n");
```

- Rechte Maustaste => Refactor => Extract => Variable
- Name = *mode* Eingeben
- *Replace all occurrences (2 occurrences)* aktivieren
- *Ok* anklicken.

Das Gleiche machen wir nun für "teardown" String in

```
buffer.append("!include -teardown . ")
.append(tearDownPathName).append("\n");
```

- Rechte Maustaste => Refactor => Extract => Variable
- Name = *mode* Eingeben
- *Replace all occurrences (2 occurrences)* aktivieren
- *Ok* anklicken.

Repeating Structure durch eine Methode ersetzen

Der folgende Code in die if-Anweisung kommt 4 mal in sehr ähnlicher Version vor:

```
if(suiteSetup != null)
 {
    WikiPagePath pagePath = crawler.getFullPath(suiteSetup);
    String pagePathName = PathParser.render(pagePath);
    buffer.append("!include -" + mode + " . ")
    .append(pagePathName).append("\n");
}
```

- Body von *if(suiteSetup != null)* markieren
- Also Code zwischen { und } markieren
- Refactor => Extract => Method..
- Name : *includePage*
- Parameter: *String mode*
- Parameter: *WikiPage page*
- Refactor => Keep Original Signature
- Do you want to replace this occurrence? => *All* bestätigen
- Dies ersetzt die ersten 3 ähnliche Strukturen durch die Methode *includePage*

Letzter Block müssen wir manuell eingeben:

```
WikiPagePath pagePath =
suiteTeardown.getPageCrawler().getFullPath(suiteTeardown);

String pagePathName = PathParser.render(pagePath);
buffer.append("!include -" + mode + " . ")
.append(pagePathName).append("\n");
```

Wird einfach durch den folgenden Methodenaufruf ersetzt:

```
includePage(mode, suiteTeardown);
```

Leider hat IntelliJ neue Version dies nicht automatisch gemacht!

To Do 4: Clean up, wenn nötig

- z.B. bessere Namen geben
- Unnötige geschwefelte Klammern entfernen
- Inline… = Variablen durch ihren Wert ersetzen
- String.format benutzten

In unserem Beispiel entfernen nun die unnötigen geschwefelten Klammern von den if-Anweisungen.

Wir können für die Hilfsmethode *includePage* private void includePage(String mode, WikiPage page) {

dann *String.format* benutzen. Also der Code

```
buffer.append("!include -" + mode + " . ")
.append(pagePathName).append("\n");
```

Durch den folgenden Code ersetzen:

```
buffer.append(String.format("!include -%s . %s\n", mode,
pagePathName));
```

To Do 5: Neu entstandene Repeating Structure Eliminieren

Der folgende Code kommt in 4 ähnlichen Versionen vor:

```
WikiPage suiteSetup = PageCrowlerImpl.getInheritedPage(
              SuiteResponder.SUITE_SETUP_NAME, wikiPage);
 if(suiteSetup != null)
        includePage(mode, suiteSetup);
```

Gleiche Technik: Konstanten Parameterizieren

- *SuiteResponder.SUITE_SETUP_NAME* markieren
- Refactor => Extract => Variable => Name = pageName

Das gleiche tun wir für:

"SetUp", "TearDown", SuiteResponder.SUITE_TEARDOWN_NAME und Nennen alle Variablen einfach: pageName

- *SetUp* markieren
- Refactor => Extract => Variable => Name = pageName
- *TearDown* markieren
- Refactor => Extract => Variable => Name = pageName
- *SuiteResponder.SUITE_TEARDOWN_NAME* markieren
- Refactor => Extract => Variable => Name = pageName

Repeating Structure in einer Methode extrahieren

Wenn nötig störende Anweisung, die zwischen zwei Zeilen von einem Repeating Structure kommen, nach oben verschieben, um das Repeating structure sauber für alle zu haben. Z.B.

```
WikiPage teardown = PageCrowlerImpl
                    .getInheritedPage(pageName, wikiPage);
String mode = "teardown";
 if(teardown != null)
   includePage(mode, teardown);
```

Die Anweisung *String mode = "teardown"* stört die Struktur, also einfach nach oben verschieben, um die zusammenhängende Struktur wieder zu bekommen:

```
String mode = "teardown";
```

```
WikiPage teardown = PageCrowlerImpl
                    .getInheritedPage(pageName, wikiPage);
 if(teardown != null)
   includePage(mode, teardown);
```

Jetzt ein Repeating Struktur markieren z.B.

```
  WikiPage suiteTeardown = PageCrowlerImpl
                    .getInheritedPage(pageName, wikiPage);

 if(suiteTeardown != null)
    includePage(mode, suiteTeardown);
```

- Rechte Maustaste => Refactor => Extract => Method...
- Name : *includeIfInherited*
- Parameter: *String mode*
- Parameter: *String pageName*
- *Refactor* betätigen
- Do you want to replace this occurance? => *All*

To Do 6: Clean up, Inline..

Wir sollten immer nachsehen, ob wir die privaten Methoden auch lesbarer und besser machen können z.B.

```
private void includePage(String mode, WikiPage suiteSetup) {
```

- Hier können wir z.B. suiteSetup zu *page* umbenennen.

Nun brauchen wir die parameterizierten Konstanten nicht mehr, das heißt, wer können Sie mit ihren Werten ersetzen und entfernen. Dafür gibt es die Inline Refactoring Pattern.

```
 String mode = "setup";
    if(includeSuiteSetup)
    {
```

```
        String pageName = SuiteResponder.SUITE_SETUP_NAME;
         includePage(mode, pageName);
    }
```

Hier können wir *mode* und *pageName* entfernen:

- Die Variable *mode* in dem Methodenaufruf *includePage* markieren
- Rechte Maustaste => Inline...
- Inline all references and remove the variable => Refactor

Das gleiche wiederholen wir für *pageName*

- Die Variable *pageName* in dem Methodenaufruf *includePage* markieren
- Rechte Maustaste => Inline...
- Inline all references and remove the variable => Refactor

Wir wiederholen den Vorgang in allen 4 Strukturen und entfernen danach die unnötigen geschweiften Klammern!

To Do 7 Operationen in If-Statments in Methoden extrahieren

Jetzt können wir die Haupt Funktionalität/Operationen in separaten Funktionen mit sehr guten aussagekräftigen Namen raus extrahieren und zwar jeweils das 'body' von der if-Anweisung:

```
if(pageDate.hastAttribute("Test").
```

Zuerst extrahieren wir das Body von der ersten if-Anweisung

```
 if(includeSuiteSetup)
   includeIfInherited(
       SuiteResponder.SUITE_SETUP_NAME, "setup");
  includeIfInherited("SetUp", "setup");
```

Also wir extrahieren das Block, welche die "setups" inkludiert in eine Funktion "includeSetups". Dafür

- das Block markieren
- RMT => Refactor => Extract => Method.... => includeSetups
- Anschliessend löschen wir überflüssige {}

Analog dazu extrahieren wir den Code, der "teardowns" inkludiert in eine Function "includeTeardowns" in dem wir den folgenden Block markieren:

```
includeIfInherited("TearDown", "teardown");
    if(includeSuiteSetup)
       includeIfInherited(
           SuiteResponder.SUITE_TEARDOWN_NAME, "teardown");
```

- das Block markieren
- RMT => Refactor => Extract => Method.... => includeTeardowns
- Anschliessend löschen wir überflüssige {}

To Do 8 StringBuffer/StringBuilder Code Eliminieren

Die StringBuffer variable wurde nur benutzt um Strings aneinander anzuhängen, wir können sie elimineren. Dafür lassen wir unsere Hilfsfunktionen String zurück liefern und sie anschliessend in invoke mit "+" verketten

1) Wir ändern die Return-Value von includeIfInherited von 'void' zu 'String' und lassen die Methode ein Exception werfen.

```
   private String includeIfInherited(...) throws Exception {
     return includePage(mode, page));
   }
```

2) Wir ändern nun auch includePage(mode, page) von void zu String

```
    private String includePage(mode, page) throws Exception {
    // content.append(String.format("!include -%s .%s\n",
    // mode, pagePathName));
       return String.format("!include -%s .%s\n", mode,
pagePathName));
    }
```

3) Wir bennen die Variable StringBuffer content in 'String content' um:

- buffer Variable markieren => RMT => Refactor => Rename => 'content'
- ändern Ihren Typen anschließend zu String.
- und passen *content = new StringBuffer();*
- Zu: *content = "" ;*

4) in includeSetups: speichern und verketten wir das Ergebnis von includeIfInherited in Content

```
   content += includeIfInherited("setup",
SuiteResponder.SUITE_SETUP_NAME);
   content +=      includeIfInherited("setup", "SetUp");
```

5) Analog in includeTeardowns: speichern wir auch das Ergebnis von includeIfInherited und verketten es in Content.

```
content += includeIfInherited("teardown", "TearDown");
 if(includeSuiteSetup)
 content += includeIfInherited("teardown",
SuiteResponder.SUITE_TEARDOWN_NAME);
```

6) Das Endergebnis in die invoke() mit "+" zusammen verketten.wir ersetzen den Code:

```
    content.append(pageData.getContent());
```

durch

```
content += pageData.getContent();
content += includeSetups();
content += includeTeardowns();
```

Dafür ändern wir den Rückgabetyp von includeTeardown von void zu String. wir definieren eine locale Variable *String teardown = "";*

```
 private String includeTeardowns() throws Exception{
  String teardowns = "";
  teardowns += includeIfInherited("teardown", "TearDown");
   if(includeSuiteSetup)
   teardowns += includeIfInherited("teardown",
SuiteResponder.SUITE_TEARDOWN_NAME);
          return teardowns;
     }
```

8) Das gleiche machen wir mit includeSetups. Sinn ist, alle kleinen Funktionen in invoke() verketten. In *includeSetups()* die lokale Variable String content definieren

- content markieren => RMT => Rename => 'setups'

9) Jetzt ist die Logik von invoke() function klar und deutlich wir können diesen Code noch weiter verfeinern, in dem wir alle 'content-altering code' innerhalb eine einzige if-statment rein tun. und die untere if-statment löschen, da sie beide gleich sind!

Anschliessend Code formatieren.

- Code markieren => Code => Reformat Code

To Do 9. Extract the Predicate of the if Statement into a Method

jetzt können wir das Prädikat, also den boolsche Ausdruck in die if-Statment in der invoke() method in eine eigne Methode extrahieren:

```
if(pageData.hasAttribute("Test")
```

- pageDate.hasAttribte("Test") markieren
- RMT => Refactor => Extract => Method....=> name: istTestPage()

ToDo 11 (Extract the body of the if Statement into a method). Also das Body von

```
 if(isTestPage()){
     content += includeSetups();
```

```
        content += pageData.getContent();
        content += includeTeardowns();
        pageData.setContent(content.toString());
    }
```

- markieren => RMT => Refactor => Extract => Method => 'surroundPageWithSetupsAndTeardowns'
- Anschließend überflüssige { } löschen

To Do 10. Rename the inner class and the invoke() function

Die Invoke Function sagt uns jetzt genau, was er tut!auch alle anderen Funktionen sagen was sie genau tun! Denn alle sind: small, simple, well-names functions

- Die innere Class umhüllt (surround) eine 'test page' mit 'setups' und 'teardowns'
- Wir nennen die innere Class TestableHtmlMaker zu 'SetupTeardownSurrounder' um
- 'TestableHtmlMaker' markieren
- RMT => Refactor => Rename => 'SetupTeardownSurrounder'

Wir nennen die Methode invoke() zu surround() um

- invoke() markieren => RMT => Refactor => Rename => 'surround'

Unser Code ist jetzt wie ein "Well-Written Prose"