

# Übungen zur Vorlesung Softwaretechnologie

-Wintersemester 2014/2015-

Dr. Günter Kniesel

## Übungsblatt 12

Zu bearbeiten bis: 31.01.2015

Bitte fangen Sie **frühzeitig** mit der Bearbeitung an, damit wir Ihnen bei Bedarf helfen können. Checken Sie die Lösungen zu den Aufgaben in Ihr Repository ein, „Erklärungen“ bitte als Textdatei.

Fragen zu Übungsaufgaben/Vorlesung können Sie auf der Mailingliste [swt-tutoren@lists.iai.uni-bonn.de](mailto:swt-tutoren@lists.iai.uni-bonn.de), bzw. [swt-vorlesung@lists.iai.uni-bonn.de](mailto:swt-vorlesung@lists.iai.uni-bonn.de) stellen.

### Aufgabe 1. *Blackbox Test* (4 Punkte)

Gegeben sei folgende Schnittstelle

```
interface DateParser {  
    java.util.Date parseDate(String input);  
}
```

und folgende Dokumentation der Methode `parseDate(String input)`:

Erzeugt ein Date-Objekt aus einem Datumsstring der Form *Tag-Monat-Jahr*.

- *Tag* und *Monat* können ein- oder zweistellig sein, evtl. mit führender 0.
- *Jahr* kann zwei- oder vierstellig sein, zweistellige Angaben beziehen sich auf das 21. Jahrhundert.
- Wenn *Monat* kleiner 1 ist, wird Januar angenommen, bei Werten über 12 wird Dezember angenommen.
- Wenn *Tag* kleiner 1 ist wird der Monatserste angenommen, bei Werten größer dem zuletzt gültigen Tag der Monatsletzte.

**Parameter:**

Der Datumsstring, der interpretiert werden soll

**Liefert zurück:**

Das interpretierte Datum (mit dem Uhrzeitwert 12:00:00) oder *null* bei ungültiger Eingabe

- Überlegen Sie, welche Äquivalenzklassen auftreten können (s. Skript, Kapitel 10a, Seite 20). Geben Sie für fünf Äquivalenzklassen (von korrekten oder falschen Eingaben) einen Eingabestring und das erwartete Methoden-Ergebnis an.
- Im Repository `ssh://git-se@git.iai.uni-bonn.de/swt2014_readonly` finden Sie das Archiv „DateParser.zip“. Darin ist in der Datei `lib/GemaltoDateParser.jar` der ByteCode einer Klasse enthalten, die das Interface `DateParser` implementiert. Im Ordner `tests` finden Sie die noch unvollständige Klasse `GemaltoDateParserTest` die die Methode `parseDate` aus der Klasse `GemaltoParser` mit *JUnit 4* testet. Vervollständigen Sie diese Tests indem Sie jede in (a) identifizierte Fehlerart in einen Testfall umsetzen.  
→ **Hinweis:** Nutzen Sie in den Tests die Hilfs-Methode `makeDate(...)`.  
→ **Bonus (2 Punkte):** Welchen Fehler hat die getestete Klasse?

## **Aufgabe 2.** *Test-Driven Development* (8 Punkte)

Da die Klasse `GemaltoParser` fehlerhaft aber nicht als Quellcode verfügbar ist, müssen Sie selbst eine korrekte Implementierung der Spezifikation aus Aufgabe 1 erstellen. Dabei nutzen Sie als Hilfe die schon erstellten Tests. Gehen Sie wie folgt vor:

- Schreiben Sie eine Klasse `MyDateParserTest`, welche von `DateParserTest` erbt, aber die `setUp`-Methode so überschreibt, dass die Variable `systemUnderTest` mit einer Instanz der Klasse `MyDateParser` initialisiert wird.
- Entwickeln Sie nun die Klasse `MyDateParser`, indem Sie nach jedem Entwicklungsschritt den JUnit-Test `MyDateParser` ausführen, bis alle Testmethoden erfolgreich sind (Zur Dokumentation Ihrer Vorgehensweise sollen Sie nach jeder Modifikation Ihre Klasse in Ihr Repository einchecken).

## **Aufgabe 3.** *Refactoring* (8 Punkte)

- a) Überlegen Sie sich welche Probleme beim Verlagern einer Methode in eine andere Klasse (Move Method Refactoring) auftreten können. Beachten Sie insbesondere auch Sichtbarkeiten und Vererbung. Beschreiben Sie mindestens 3 verschiedene Probleme.
- b) Führen Sie mit Eclipse auf einem bestehenden Programm drei verschiedene automatisierte Refactorings durch. Beschreiben Sie was Sie tun und was das System tut.

**Tipp:** Wenn Sie im Java-Editor von Eclipse mit dem Cursor auf eine Klasse / Variable / Methode gehen und die rechte Maustaste drücken, wird Ihnen unter „Refactor“ ein Katalog möglicher Refactorings angeboten. Ditto im „Refactoring“-Menue.