Fehler vermeiden (JUnit)

Lernziele

- Die Studierenden können JUnit-Tests schreiben.
- Die Studierenden kennen die `assertEquals`-Methode von JUnit und können diese verwenden.
- Die Studierenden kennen den Zugriffsmodifikator `package-private`.

Aufgabe 1

Forken Sie für diese Aufgabe das Projekt
https://github.zhaw.ch/prog1-kurs/07_Praktikum_Worthaeufigkeitsanalyse.

Nutzen Sie **eine IDE** um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Sie haben in einem vorhergehenden Praktikum ein Programm zum Analysieren der Worthäufigkeit in Texten geschrieben. Das Projekt 07_Praktikum_Worthaeufigkeitsanalyse beinhaltet eine mögliche Lösung für diese Aufgabe, die nun getestet werden soll.

a) Schreiben Sie einen Test für die Methode `entferneSatzzeichen`. Die Methode soll folgende Funktionalitäten erfüllen:

   i) Am Anfang und am Ende eines Wortes werden die Satzzeichen (`. , ? ! " : ;`) entfernt.

   ii) Ist ein Satzzeichen in einem Wort, z.B. bei Abkürzungen, wird dieses nicht entfernt.

Hinweis: Wenn Sie jetzt einen JUnit-Test schreiben wollen, um die Methode `entferneSatzzeichen` zu testen, dann können Sie dies nicht tun, da der Zugriffsmodifikator `private` ist. Diese Situation trifft oft auf, wenn nachträglich zu bestehendem Code JUnit-Tests geschrieben werden sollen. Welche Möglichkeiten haben wir, damit die Methode trotzdem getestet werden kann?

- Die erste Möglichkeit ist, den Zugriffsmodifikator von `private` auf `public` zu ändern. Meistens ist dies aber nicht gewünscht, da dies die Schnittstelle der Klasse verändert.
- Die zweite Möglichkeit arbeitet mit Vererbung und dem Zugriffsmodifikator `protected`. Da wir das Konzept der Vererbung noch nicht behandelt haben, verschieben wir die Diskussion dieser Möglichkeit auf später.
- Eine dritte Möglichkeit, welche wir für diese Aufgabe benutzen wollen, ist am Ende dieses Dokumentes, im Anhang *Appendix Zugriffsmodifikator,* beschrieben. Lesen Sie diesen Anhang, bevor Sie die Aufgabe lösen.

  Zum Thema testen von privaten Methoden herrschen kontroverse Meinungen. Eine grobe Übersicht über die Argumente dafür und dagegen sind hier zu finden:
  https://bytebaker.wordpress.com/2009/02/13/testing-private-methods-or-not/
  https://jesseduffield.com/Testing-Private-Methods/

b) Schreiben Sie einen Test für die Methode `verarbeiteText`. Diese Methode hat keinen Rückgabewert. Um zu prüfen ob die Klasse nach dem Aufruf von dieser Methode den korrekten

Zustand hat, müssen Sie in der zu testenden Klasse Hilfsmethoden einfügen. Überlegen Sie sich, welche Fälle zu testen sind (insbesondere ob auch Wiederholungen eines Worts richtig erkannt werden) und welches Ergebnis Sie erwarten. Überlegen Sie sich auch einen negativen Test. Folgende Funktionalität muss gewährleistet sein:

i) Gross- und Kleinschreibung der Wörter soll nicht beachtet werden.

ii) Es muss die Häufigkeit von jedem Wort genau erfasst werden. Die Häufigkeit eines Wortes ist grösser oder gleich eins.

iii) Ein Text ist durch ein oder mehr Zeichen definiert. Der leere String gilt nicht als Text.

c) Korrigieren Sie den Code so, dass alle Tests ohne Fehler durchlaufen.

Aufgabe 2

Forken Sie für diese Aufgabe das Projekt
https://github.zhaw.ch/prog1-kurs/07_Praktikum_Notenprogramm.
Nutzen Sie **eine IDE** um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten. Das Projekt 07_Praktikum_Notenprogramm beinhaltet eine mögliche Lösung für ein früheres Praktikum. Diese Lösung soll nun hier getestet werden.

a) In der Klasse `Pruefungsverwaltung` gibt es die Methode `rundeAufHalbeNote`, diese Methode soll folgende Anforderungen erfüllen:

- Noten mit beliebig grosser Genauigkeit sollen auf halbe Noten gerundet werden. Hierbei gilt, dass .24 und .74 ab- und .25 und .75 aufgerundet werden.

Testen Sie, ob diese Methode die Noten korrekt rundet. Überlegen Sie sich dazu zuerst die möglichen Äquivalenzklassen. Schreiben Sie anschliessend die dazugehörigen Unit-Tests.

b) In der Klasse `ZufaelligeNotengebung` gibt es die Methode `generiereZufaelligePruefugsnote`. Diese Methode liefert (pseudo-)zufällig verteilte Noten von 1 bis 6 zurück. Solche Methoden sind schwierig zu testen, da der Rückgabewert nicht vorhersehbar ist. Überlegen Sie sich, wie Sie trotzdem einen Test schreiben können, welcher gewährleistet, dass die Methode die korrekten Prüfungsnoten von 1 bis 6 generiert.

Überlegen Sie sich mit Hilfe der JavaDoc, warum die Note welche von `generiereZufaelligePruefugsnote` nie genau 6.0 erreicht.

c) Korrigieren Sie den Code so, dass alle Tests ohne Fehler durchlaufen.

Appendix Zugriffsmodifikator

Sie kennen bereits die Zugriffsmodifikatoren `private` und `public`. Der Zugriffsmodifikator `public` erlaubt einen globalen Zugriff auf Methoden und Variablen – es gibt also keinerlei Einschränkung um auf eine solche `public` Methode oder Variable zuzugreifen.

Der `private`-Zugriffsmodifikator hingegen ist der restriktivste, da er nur einen Zugriff innerhalb der gleichen Klasse erlaubt. Auch eine Subklasse hat keinen Zugriff.

Sie lernen jetzt einen dritten Zugriffsmodifikator kennen. In der Vorgabe für die Aufgabe 1 finden Sie in Worthaeufigkeitsanalyse.java die Methode `entferneSatzzeichen`. Diese Methode hat folgende Signatur:

```
private String entferneSatzzeichen (String wort)
```

Wenn Sie jetzt einen JUnit-Test schreiben wollen, um diese Methode zu testen, dann können Sie dies nicht tun, da der Zugriffsmodifikator `private` ist. Eine (schlechte) Möglichkeit wäre, den Zugriffsmodifikator von `private` auf `public` zu ändern. Der Nachteil dabei ist, dass dann diese Methode global aufgerufen werden kann. Eine elegantere Lösung ist, einen Zugriffsmodifikator zu wählen, welcher nicht so restriktiv wie `private` ist, aber auch nicht so global wie `public` – sozusagen ein `private`-Zugriffsmodifikator, welcher zusätzlich den Zugriff für den JUnit Test erlaubt. Dies ist möglich, wenn man *keinen* Zugriffsmodifikator definiert, dann ist der Zugriff innerhalb desselben Packages erlaubt:

```
String entferneSatzzeichen (String wort)
```

Da in diesem Praktikum die Testklasse und die zu testenden Klassen im gleichen Package liegen (dem sog. default-Package, da wir noch keine expliziten Packages verwenden), kann die Testklasse deshalb auf die Methode zugreifen.

Die untere Tabelle zeigt eine Übersicht, woher in Abhängigkeit des Zugriffmodifikators die Verwendung einer Methode (oder eines Datenfelds) möglich ist.

Zugriffsmodifikator	Klasse	Package	Sub-Klasse	Global
`public`	Ja	Ja	Ja	Ja
-	Ja	Ja	Nein	Nein
`private`	Ja	Nein	Nein	Nein

Hinweis: Es gibt verschiedene Namen für den neuen Zugriffsmodifikator. Teilweise wird er Package-, Kein- oder auch Friendly-Zugriffsmodifikator genannt. In der Java Language Specification wird «**Package-Access**» verwendet, im unten aufgeführten Tutorial der in der Praxis oft verwendete Begriff «**package-private**».

Studieren Sie die folgende Website von Oracle, welche alle Zugriffsmodifikatoren und deren Eigenschaften im Detail beschreibt:
https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html