Aufgabe Craps mit Mockito



Einleitung zu Craps

Es liegt eine Implementierung von Craps vor.

Sie testen nun diese Implementierung gemäss den Regeln von Craps. Dazu verwenden Sie Unit-Tests, die möglichst vielfältig die Spielverläufe testen.

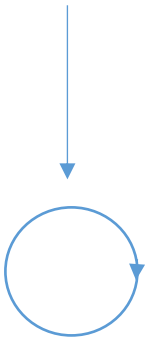
Das Regelwerk von Craps

Der Spieler würfelt **immer mit beiden Würfeln** gleichzeitig gegen eine Bande des Spielfeldes.

Die Würfel-**Werte werden zusammengezählt**, die Summe entscheidet über gewinnen oder verlieren:

Würfelt er im 1. Wurf:

- eine 7 oder 11, so gewinnt er.
- eine 2, 3 oder 12, so verliert er.
- Bei einer anderen Summe darf er nochmals würfeln.



Würfelt er im Folge-Wurf:

- eine 7, so verliert er
- die gleiche Summe wie im direkten Wurf zuvor, so gewinnt er.
- Bei einer anderen Summe als im direkten Wurf zuvor, so darf er nochmals würfeln.

Die Zuschauer wetten nun auf die verschiedensten Varianten, wie das Spiel verlaufen wird.

Aufgabe 1 Unittests zu Craps

Sie müssen möglichst alle Möglichkeiten aus dem Regelwerk mittels Unittests testen:

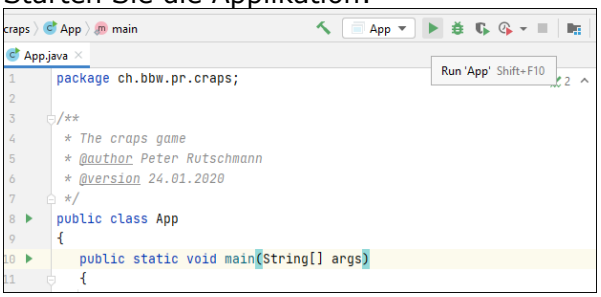
- Je ein Unit Test für **die Möglichkeiten** (gewinnen und verlieren) des 1. Wurfes
- Je ein Unit Test für **die Möglichkeiten** (gewinnen und verlieren) des ersten Folge Wurfes.
- Je ein Unit Test für **die Möglichkeiten** (gewinnen und verlieren) des zweiten Folge Wurfes.
- Je ein Unit Test für **die Möglichkeiten** (gewinnen und verlieren) des vierten Folge Wurfes.

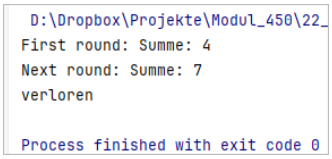
⇨ Das Vorgehen ist auf der nächsten Seite beschrieben.

Vorgehen

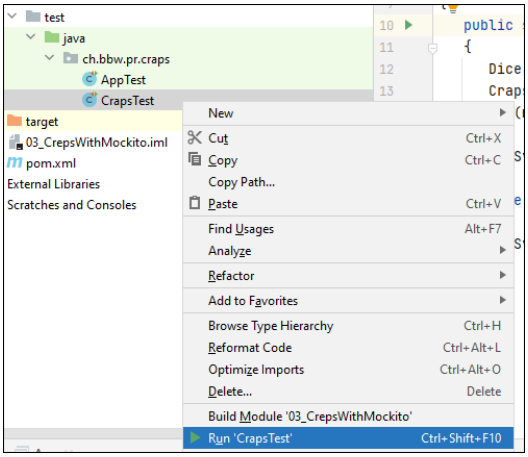
1. Öffnen Sie das Projekt aus der Vorgabe *CrapsWithMockitoVorgabe.zip*.
   Es enthält die vorgegeben Klassen *App*, *Craps* und *Dice*, sowie einen ersten Test in der Klasse *CrapsTest*.
2. Im POM.xml sind bereits die Dependencies für Unittests mit JUnit 4 und Mockito enthalten.
   Google: Maven dependency junit 4
   Google: Maven dependency mockito

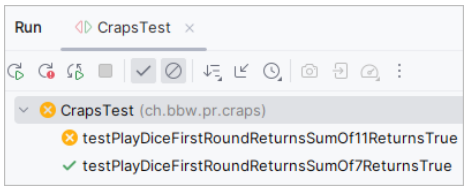
   (Sie können das im pom.xml der Vorlage kontrollieren.)
3. Starten Sie die Applikation:





4. Starten Sie den Test





5. Ergänzen Sie die existierenden Tests.
   Wenden Sie Mockito zum Mocken an! Beispiel? Siehe 22_2_MockitoTheorie.pdf Seite 2
6. Ergänzen Sie weitere Tests, um die möglichen Spielzüge und die erwarteten Ergebnisse zu testen.

   Welche Namen gibt man den Tests?

> ⇨ Es soll gleich klar sein, um was es geht:
>
> - Welche Methode wird getestet? play
> - Was gibt Dice in welcher Runde zurück? DiceFirstRoundReturnsSum7
> - Was ist das Resultat der Methode? ReturnsTrue
>
> Damit ist es später einfacher einen fehlgeschlagenen Test zu identifizieren und zu verstehen, was eigentlich getestet wurde.

⇨ Es geht noch weiter !!

7. Starten Sie die Unittests und schauen Sie, ob der Test ok ist oder fehlschlägt.
   ACHTUNG: Die Implementierung hat Fehler. Es ist somit zu erwarten, dass einige Tests fehlschlagen werden.
   Falls Unit Tests fehlschlagen, **kontrollieren und verbessern Sie die Implementation von *play***.

   WICHTIG:
   Immer zuerst den Test schreiben, erst dann Implementation anpassen… falls nötig.
   ➔ Test Drive Development 😊

Weitere Aufgaben:

8. Nicht alle Fehler findet man nur durch Test *Soll-Ergebnis versus IST-Ergebnis*.
   Es gibt weitere Aspekte, zum Beispiel, ob wirklich so oft wie gefordert gewürfelt wird, um das Ergebnis zu erhalten.
   Es könnte ja auch sein, dass das Ergebnis mit weniger oder nur mit mehr Würfen als gefordert trotzdem erreicht wird.

   Ergänzen Sie all Ihre Tests deshalb um die Überprüfung der "number of invocations"
   Google: JavaDoc Mockito.
   - 4. Verifying exact number of invocations…

   Nachschlagen mit Google, lesen und ausprobieren.

9. Zum Abschluss, wenn alle Tests erfolgreich durch-laufen:
   Manipulieren Sie die Implementation von *play* und kontrollieren Sie, ob die Unit Tests fehlschlagen würden.

Aufgabe 2 JavaDoc zu Mockito

Das Mockito Framework bietet noch viele andere Möglichkeiten, das Verhalten eines Mocks zu beeinflussen, zu überwachen ….

Lesen Sie dazu die JavaDoc zu Mockito.
Google: JavaDoc Mockito.

Probieren Sie die beschriebenen Beispiele praktisch aus.

Interessante Themen:

- 1. Let’s verify some behaviour: -> Wurde eine Methode des Mocks aufgerufen.
- 2. How about some stubbing?
- 3. Argument matchers
- 4. Verifying exact number of invocations…
- 6. Verification in order
- 7. Make sure interaction(s) never happened on mock

Aufgabe 3 Debugger, Unittest und Mockito

Untersuchen Sie die Möglichkeit einen Unittest mit Hilfe des Debuggers zu untersuchen.
Ist das auch mit einem Mock möglich?