

Test-Driven Development

Aufgabe "Test-Driven Development", Modul 450

Die folgenden Aufgaben beziehen sich auf die Beispiele **WordsGame** (die eigentlich Logik), **WordsGame.Demo** (das interaktive Demoprogramm für die Spiellogik) und **WordsGame.Test** (das xUnit-Test-Projekt, welches noch keine Tests enthält).

Das Spiel funktioniert folgendermaßen:

1. Es wird eine Reihe von Wörtern (Substantive) mit sechs bis acht Buchstaben in einer Textdatei zur Verfügung gestellt (`words.tx`).
2. Beim Starten des Spiels wird eines dieser Wörter per Zufallsverfahren ausgewählt.
 - z.B. ZWIEBACK
3. Das Buchstaben des Wortes werden zufällig durcheinandergebracht.
 - z.B. KBWIZAEK
4. Der Spieler muss das Original erraten.
 - Errät der Spieler das richtige Wort, erhält er pro Buchstabe einen Punkt.
 - Gibt der Spieler ein anderes Wort ein, erhält er keine Punkte.

Dein Auftrag ist es, die Spiellogik gemäß dem Vorgehen des Test-Driven Developments (TDD) fertig zu entwickeln. Zur einfacheren Vergleichbarkeit werden die Wörter jeweils in Großbuchstaben umgewandelt. (Würde nur der Anfangsbuchstaben großgeschrieben, würde das Spiel dadurch zu einfach werden.)

1. Anweisungen

Löse die untenstehenden Aufgaben. Halte dich dabei an das Vorgehen von Test-Driven Development.

1. Schreibe keinen neuen Produktivcode, solange kein Test scheitert.
2. Schreibe nur so viel Testcode, bis der Test scheitert.
3. Schreibe nur so viel Produktivcode, bis der Test durchläuft.

2. Aufgabe 1 (vorgegeben): Wort durcheinanderbringen

Die Klasse `Utils` im `WordsGame`-Projekt verfügt über eine statische Methode namens `Scramble`, welche einen `String` erwartet und einen anderen `String` zurückgibt. Die Methode soll den `String` durcheinanderbringen.

Implementiere die Logik selber (nur mithilfe von Arrays, Schleifen, Zufallszahlen und anderen grundlegenden C#-Sprachkonstrukten), indem du nacheinander die folgenden Fälle als Tests und dann im Produktivcode implementierst:

1. Der leere `String` `""` wird zu einem leeren `String` `""`.
2. Ein `String` bestehend aus einem Zeichen (z.B. `"X"`) wird zu einem `String` bestehend aus dem gleichen Zeichen.
3. Ein `String` bestehend zwei Zeichen (z.B. `"XY"`) wird zu einem `String` bestehend aus den gleichen Zeichen, die aber in einer anderen Reihenfolge angeordnet sind (`"YX"`).
 - Aus dem zufälligen Durcheinanderbringen des `Strings` kann durchaus der ursprüngliche `String` das Ergebnis sein.
 - In diesem Fall muss die Methode `Scramble` den Vorgang wiederholen, bis ein anderes Ergebnis erreicht wird.
4. Ein `String` bestehend aus drei oder mehr Zeichen (z.B. `"XYZ"`) wird zu einem gleichlangen `String` mit unterschiedlicher Reihenfolge (z.B. `"ZYX"`).
 - Es gelten die gleichen Regeln wie bei `Strings` bestehend aus zwei Zeichen.
 - Eigentlich handelt es sich hierbei um die gleichen Fälle, zumal die Unterscheidung in 0, 1 und n hier genügt.

Führe immer alle Tests durch, damit durch eine hinzugefügte Funktionalität keine bestehende Funktionalität beeinträchtigt wird.

3. Aufgabe 2 (teilweise vorgegeben): Spiel implementieren

Das Interface `IWordsGame` definiert zwei Methoden, auf Basis derer die Spiellogik implementiert werden soll:

1. `string Start(string word)`: Die Methode nimmt das Originalwort als `String` entgegen und gibt den durcheinandergemischten `String` zurück, der dann dem Spieler angezeigt werden kann.
2. `int Grade(string solution)`: Die Methode nimmt den Lösungsvorschlag als `String` entgegen und bewertet ihn, indem eine Punktzahl zurückgegeben wird.

Damit die Spiellogik auf Basis dieses Interfaces implementiert werden kann, muss das Originalwort als Eigenschaft der Klasse zwischengespeichert werden.

Die Klasse `WordsGame` implementiert das Interface `IWordsGame`, ist aber noch nicht fertig implementiert. Führe die Implementierung nach TDD-Methode zu Ende. Beachte dabei folgende Regeln:

1. Das Interface darf nicht angepasst werden.
2. Die Methode `Start` soll die Methode `Utils.Scramble` aus Aufgabe 1 verwenden.
3. Werden die Methoden `Start` und `Grade` mit dem gleichen `String` aufgerufen, soll letztere die Anzahl Zeichen dieses `Strings` als Punktezahl zurückgeben.
4. Unterscheiden sich die `Strings` hingegen, erhält der Spieler null Punkte.

Teste anschließend das Spiel interaktiv (siehe `README.md` im Repository).

4. Aufgabe 3 (frei): Bewertung erweitern

Schreibe eine alternative Implementierung, welche wiederum IWordsGame implementiert. Die Punktezahl soll gemäß Scrabble-Regeln vergeben werden:

Punkte	Buchstaben
1	A, E, I, L, N, O, R, S, T, U
2	D, G
3	B, C, M, P
4	F, H, V, W, Y
5	K
8	J, X
10	Q, Z

Am besten implementierst du zuerst eine Hilfsmethode in der Utils-Klasse nach TDD-Vorgehen, welche die Punktevergabe implementiert. Zum Testen der neuen Spiellogik sind dann nur noch wenige Testfälle nötig.