



## Aufgabe 0.4

Geben Sie das Lösungswort des Quiz aus der Lernnotiz an.

## Aufgabe 5

Gegeben sei das von der Webseite erhältliche und in der Veranstaltung vorgestellte Programm im Projekt `sqmAufgabeEigenerTester`, mit dem eine Annotation `@MeinTest` zur Beschreibung und Ausführung von eigenen Testfällen genutzt wird.

- Erweitern Sie die Verarbeitung der Annotation um ein optionales Attribut erlaubt `@MeinTest(erlaubt={ArithmeticException.class})`, das eine Liste (Array) von Class-Objekten von Exceptions enthält. Wirft die so annotierte Methode eine Exception, ist dann zu prüfen, ob sie sich in der Liste befindet und der Test damit in Ordnung ist, oder nicht. Bei nicht erlaubten Exceptions ist dies zu dokumentieren. Für den hier weiterentwickelten Prototypen reicht eine Ausgabe. Beachten Sie, dass die Ausführung bei einer Exception zunächst immer ein Objekt vom Typ `InvocationTargetException` wirft und aus diesem mit der Methode `getCause()` die eigentliche Exception erhalten werden kann.
- Erweitern Sie die Verarbeitung der Annotation um ein optionales Attribut `datei` `@MeinTest(datei="tests.txt")`, in dem der Name einer Datei (String) mit trennzeichenseparierten Testdaten steht. Dabei steht nach einer Kopfzeile in jeder Zeile der Datei eine Sammlung von Testdaten. Die zugehörigen Tests haben Parameter, die zu dieser Datei passen müssen, was Sie natürlich prüfen. Sieht die Datei mit den Testdaten wie folgt aus (die Excel- Datei wurde mit „CSV (Trennzeichen getrennt)“ abgespeichert, das dort genutzte Trennzeichen ist „;“),

	A	B	C	D	E
1	Nutzer1	Nutzer2	Wert	Ergebnis	
2	Blubb	Bing		2 false	
3	Blubb2	Blubb		2 true	
4	Bing123	Bing	123	true	
5	Bing123	Bing1	23	true	
6	Bing123	Bing12	3	true	
7	Bing123	Bing123	0	false	
8	Bing	Bum	Bang	false	
9	Zuviele	Daten	1	false	inDieserZeile
10	ZuWenige	Zeilen			
11	Bing123	Bing		1	
12		1234	12	34	true
13		1234	12	34	Hallo
14	FalscherTest0	FalscherTest	0	false	
15					

und sei eine Methode der folgenden Form gegeben,

```
@MeinTest(datei="Testdaten.csv")
public void testMitDatei(String s1, String s2, int i, boolean b)
    throws Exception{
    if (s1.equals(s2 + i) != b) {
```



```
        throw new Exception(s1 + " eq " + s2 + i + " sollte "  
                               + (b ? "wahr sein" : "falsch sein"));  
    }  
}
```

könnte ein Test der für jeden Datensatz ausgeführt wird, zu folgenden Beispielausgaben führen. Fehlerhafte Daten sind zu dokumentieren und führen zu keinem Abbruch.

```
Testdaten.csv: Problem bei Umwandlung von [Bing, Bum, Bang,  
false] in Typen [class java.lang.String, class java.lang.String,  
int, boolean]: java.lang.NumberFormatException: For input  
string: "Bang"  
Testdaten.csv: erwartet wurden 4 Parameter, gefunden wurden 5:  
[Zuviele, Daten, 1, false, inDieserZeile]  
Testdaten.csv: erwartet wurden 4 Parameter, gefunden wurden 2:  
[ZuWenige, Zeilen]  
testMitDatei_0: [Blubb, Bing, 2, false]: ok  
testMitDatei_1: [Blubb2, Blubb, 2, true]: ok  
testMitDatei_2: [Bing123, Bing, 123, true]: ok  
testMitDatei_3: [Bing123, Bing1, 23, true]: ok  
testMitDatei_4: [Bing123, Bing12, 3, true]: ok  
testMitDatei_5: [Bing123, Bing123, 0, false]: ok  
testMitDatei_6: [Bing123, Bing, 1, false]: ok  
testMitDatei_7: [1234, 12, 34, true]: ok  
testMitDatei_8: [1234, 12, 34, false]: Exception : 1234 eq 1234  
sollte falsch sein  
testMitDatei_9: [FalscherTest0, FalscherTest, 0, false]:  
Exception : FalscherTest0 eq FalscherTest0 sollte falsch sein
```

Sie dürfen vereinfachend annehmen, dass nur die elementaren Datentypen int, double und boolean sowie String vorkommen können. Da Java nicht dynamisch typisiert ist, werden Sie darauf prüfen müssen.

Die Datei Beispiel.java im Projekt enthält einige annotierte Methoden, die alle ausgeführt werden sollen, etwaige Fehler sind zu dokumentieren.

Sie dürfen den Prototypen natürlich optimieren und z. B. kürzere Methoden nutzen.