Exercise 9

1 Data-driven testing of a dice game implementation

a)

For K2 (c) we specified which Exception is expected to be thrown and what message it should have.
The corresponding test is named `changeDiceStateWithValidDiceNot1Or5` and runs successfully.

For K3 (a) we didn't know which Exception should be thrown so we asserted that a SelectionException should be thrown and we also specified what the message should/could look like.
The corresponding test is named `changeDiceStateWithValidDice1Or5` and fails. This is because the code does not throw an exception as is expected by the specification of the test scenario. We used a parameterized test to cover both scenarios: Dice has value 1 or dice has value 5.

To make the second test run successfully as well the implementation of the dicegame needs to be altered. The method `changeDiceState` does not differentiate between dice which were fixed in this round and those that were fixed in a previous round. Therefore it allows permanently fixed dice (from previous rounds) to be unfixed. Instead an exception should be thrown, whenever someone tries to alter the state of a permanently fixed dice.

b)

We implemented the test cases K3 (b) and K3 (c). We implemented the test cases as parameterized tests. Therefore our paths are the following:

K1 b → K2 a → K3 b
K1 b → K2 b → K3 b

K1 b → K2 a → K3 c
K1 b → K2 b → K3 c

2 Interface Testing

The created tests can be found in the test module at
src/test/java/org/jabref/model/search/matchers

The **SearchMatcherTest** is the abstract test class of the SearchMatcher interface.
It is extended by:

- **AndMatcherTest**
- **OrMatcherTest**
- **NotMatcherTest**
- **NotAndMatcherTest** (combinations)
- **NotOrMatcherTest** (combinations)

Note for the tutor:

There was some confusion around whether we should build the interface test for **SearchMatcher** or **MatcherSet**. We ultimately decided to go with **SearchMatcher**, because **NotMatcher** implements **SearchMatcher** (but does not extend **MatcherSet**).