SPECIFICATION BASED TESTING TECHNIQUES EXERCISES

DECISION TABLES

EXERCISE: LEAP YEARS

Make a decision table for leap years

Leap year: Most years that are evenly divisible by 4 are leap years.
An exception to this rule is, that years that are evenly divisible by 100 are not leap years, unless they are also evenly divisible by 400, in which case they are leap years.

Conditions								
Actions								

EXERCISE: CREDIT CARD DISCOUNT

When a customer is getting a credit card, the customer will then receive a discount percentage, based on being a new or an existing customer and having a loyalty card or a coupon

- 15% discount on all purchases for a new customer
- 10% discount on all purchases for an existing customer with a loyalty card
- 20% discount on all purchases for an existing customer with a coupon

1. Create a decision table for credit card discount conditions and actions

Conditions								
Actions								

2. Implement code and tests for the credit card discount decision table

STATE TRANSITION MODELS

EXERCISE: ACCOUNT – GETINTEREST / DEPOSIT / WITHDRAW

A savings account in a bank earns a different rate of interest depending on the balance in the account
A balance in the range \$0 up to \$100 has a 3% interest rate
A balance over \$100 and up to \$1000 has a 5% interest rate
A balance of \$1000 and over have a 7% interest rate

1. Create a state transition model for Account

2. Implement Account class with getInterest, deposit and withdraw methods
3. Create tests covering multiple states and transitions

REPEATED TESTS

1. Create repeated tests for the method getInterest in the Account class that tests the different values
2. Create repeated tests for the method getDiscount in the Credit Card class that tests the different values

PARAMETERIZED TESTS

1. Create parameterized tests for the method getInterest in the Account class that tests the different values by using the @ValueSource annotation
2. Create parameterized tests for the method getDiscount in the Credit Card class that tests the different values by using the @ValueSource annotation
3. Create parameterized tests for the method getDiscount in the Credit Card class that tests the different values by using the @CsvSource annotation
4. Create parameterized tests for the method getDiscount in the Credit Card class that tests the different values by using the @CsvFileSource annotation
5. Create parameterized tests for the method getInterest in the Account class that tests the different values by using the @MethodSource annotation
6. Create parameterized tests for the method getDiscount in the Credit Card class that tests the different combinations by using the @MethodSource annotation

DYNAMIC TESTS

1. Create dynamic tests for the method getInterest in the Account class that tests the different values by using the @TestFactory annotation
2. Create dynamic tests for the method getDiscount in the Credit Card class that tests the different combinations by using the @TestFactory annotation

HAMCREST

1. Rewrite some of the tests to use Hamcrest matchers instead of JUnit asserts