SPECIFICATION BASED TESTING TECHNIQUES ASSIGNMENT

IMPLEMENT AND TEST BANK SYSTEM

1. Implement a minimal basic bank system with Account / Customer / Credit card / ATM classes and methods supporting the following features…
    - Creation of customers with accounts
    - Addition of credit cards to customers
    - Manipulation of credit cards by ATMs
        - Insert credit card
        - Extract credit card
        - Deposit amount
        - Withdraw amount
        - Purchase with discount
        - Show balance
        - Show monthly interest rate
        - Show yearly interest rate

    The exact details of the bank system is not predetermined and therefore open for interpretation, with the exception of the creation of credit cards and calculation of monthly interest rates.

    Credit cards should be created the following way…
    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
    - If you are a new customer opening a credit card account, you will get a 15% discount on all your purchases today
    - If you are an existing customer and you hold a loyalty card, you get a 10% discount
    - If you have a coupon, you can get 20% off today (but it can't be used with the 'new customer' discount)

    Monthly interest rate should be calculated the following way…
    - 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

2. Test account.Account.getMonthlyInterest() and creditcard.CreditCard.getDiscount() methods sufficiently

3. Incorporate both a repeated test, a parameterized test and a dynamic test for the creditcard.CreditCard.getDiscount() method

4. Create 4 different versions of the parameterized test, using the annotations @ValueSource, @CSVSource, @CSVFileSource and @MethodSource for the test data in each version

5. Use Hamcrest matchers throughout the tests instead of JUnit asserts

6. Select a part of the bank system and develop it a bit further, then set up some additional repeated tests, parameterized tests or dynamic tests for it with Hamcrest matchers

7. Document how equivalence partitions, boundary values, decision tables and state transition models have been used and applied in the creation of the data-driven tests