|  |  |  |  |
| --- | --- | --- | --- |
| **Java class/method name being tested: MoneyMarket.java/monthlyInterest()** | | | |
| Test Case # | Requirement | Test Description and Input Data | Expected result/output |
| 1 | Loyal Money Market Accounts must have an annual interest rate of 0.95%. | * Create an instance of MoneyMarket and an initial deposit of $2500. * Call monthlyInterest() * Test data: 2500 | 2500 \* (((0.95) / 100.0) / 12.0) = 1.98 |
| 2 | Non-loyal Money Market Accounts must have an annual interest rate of 0.8%. | * Create an instance of MoneyMarket and an initial deposit of $2500. * Withdraw $1000 from the account to make balance $1500 which is less than $2500 so it becomes non-loyal. * Call monthlyInterest() * Test data: 1500 | 1500 \* (((0.8) / 100.0) / 12.0) = 1 |
| 3 | When a non-loyal account becomes loyal, the annual interest rate must change from 0.8% to 0.95%. | * Create an instance of MoneyMarket with initial deposit of $2500. * Withdraw $1000 so it becomes non-loyal. * Call monthlyInterest() on the non-loyal account * Deposit $1000 back so it becomes loyal again * Call monthlyInterest() on loyal account * Test data: $1500 and $2500 | Non-loyal: 1500 \* (((0.8) / 100.0) / 12.0) = 1  Loyal: 2500 \* (((0.95) / 100.0) / 12.0) = 1.98 |
| 4 | When a loyal account becomes non-loyal, annual interest rate must change from 0.95% to 0.8%. | * Create an instance of MoneyMarket with initial deposit of $2500 so it is loyal by default * Call monthlyInterest() * Withdraw $2000 so that the balance falls to $500 and the account becomes non-loyal. * Call monthlyInterest() * Test data: $2500 and $500 | Loyal: 2500 \* ((0.95 / 100) / 12) = 1.98  Non-loyal: 500 \* (0.8 / 100) / 12 = 0.33 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Java class/method being tested: AccountDatabase.java/close()** | | | |
| Test Case # | Requirement | Test Description and Input Data | Expected Result/Output |
| 1 | If an existing checking account is open, calling close() will close the account in the database. | * Create an instance of Account with actual type as checking. * Call the open() method to open the created Account * Call the close() method * Input Data:   Account type: Checking  First Name: “John”  Last Name: “Doe”  Balance: $2500  Closed: False | Expected Output: True  Expected Result:  closed = true  Balance = 0 |
| 2 | If an existing college checking account is open, calling close() will close the account in the database. | * Create an instance of Account with actual type as college checking * Call open() method to open the created account * Call the close() method * Input Data:   Account type: College Checking  First Name: “John”  Last Name: “Doe”  Balance: $2500  Closed: False  Campus Code: 0 | Expected Output: True  Expected Result:  Closed = true  Balance = 0  Campus Code = 0 |
| 3 | If an existing savings account is open, calling close() will close the account in the database. | * Create an instance of Account with actual type as savings * Call the open() method to open the created Account * Call the close() method * Input Data:   Account type: Saving  First Name: “John”  Last Name: “Doe”  Balance: $2500  Closed: False  Loyal: true | Expected Output: True  Expected Result:  Closed = true  Balance = 0  Loyal = false |
| 4 | If an existing money market account is open, calling close() will close the account in the database. | * Create an instance of Account with actual type as money market * Call the open() method to open the created Account * Call the close() method * Input Data:   Account type: MM  First Name: “John”  Last Name: “Doe”  Balance: $2500  Closed: False  Loyal: true | Expected Output: True  Expected Result:  Closed = true  Balance = 0  Loyal = false |
| 5 | If an account does not exist in the database, you cannot close that account. | * Create an instance of Account through a subclass * Do not call open() on that account * Call close() * Input Data:   Account type: Checking  First Name: “John”  Last Name: “Doe”  Balance: $2500  Closed = false | Expected Output: False |
| 6 | If an existing account in the database is already closed, you cannot close that account again | * Create an instance of Account through any subclass * Open() and close() that account * Then try to close that same account again * Input Data:   Account type: Checking  First Name: “John”  Last Name: “Doe”  Balance: $2500  Closed: False | Expected Output: False |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Java class/method being tested: AccountDatabase.java/open() method** | | | |
| **Test Case** | **Requirement** | **Test Description and Input Data** | **Expected Result/Output** |
| 1 | If the account being added is not in the database, it must be added and the output must be true. | * Create an account that is not in the database, then call the open() method on it * Test data: * Customer: John Doe, 01/12/1979 * Checking account with $1000 balance | True |
| 2 | If the account is already in the database but it is closed, then it should be reopened and the output must be true. | * Add a closed account to the database, then call the open() method on it * Test data: * Customer: Jane Doe, 05/23/1998 * Savings account with $4000 balance, “closed” Boolean set to true | True |
| 3 | If the account being opened is already in the database and it is already open, then nothing should happen and the output must be false. | * Add an account to the database, then attempt to add that same account again. * Test data: * Customer = John Doe, 04/05/1985 * MoneyMarket account with $3000 balance | False |

|  |  |  |  |
| --- | --- | --- | --- |
| **Java class/method being tested: Data.java/isValid() method** | | | |
| **Test Case** | **Requirement** | **Test Description and Input Data** | **Expected Result/Output** |
| 1 | The valid range for the month shall be 1-12 | * Create an instance of Date with valid date and year, but the month is 0. * Test data: “0/12/2022” | False |
| 2 | The valid range for the month shall be 1-12 | * Create an instance of Date with valid date and year, but the month is greater than 12. * Test data: “13/12/2022” | False |
| 3 | The valid range for the day shall be 1-31 for the longer months | * Create an instance of Date with a valid long month and year, but the day is greater than 31. * Test data: “1/32/2022” | False |
| 4 | The valid range for the day shall be 1-31 for the longer months | * Create an instance of Date with valid long month and year, and the date is 31 (edge case). * Test data: “7/31/2022” | True |
| 5 | The valid range for the day in February shall be 1-28 on a non-leap year | * Create an instance of Date with month 2 and valid year, and the date is 29. * Test data: “2/29/2022” | False |
| 6 | The valid range for the day in February shall be 1-28 on a non-leap year | * Create an instance of Date with month 2 and valid year, and the date is 28. * Test data: “2/28/2022” | True |
| 7 | February 29th shall only be valid on a leap year. | * Create an instance of Date with month 2, day 29, and a valid leap year. * Test data: “2/29/2020” | True |
| 8 | February 29th shall only be valid on a leap year. | * Create an instance of Date with month 2, day 29, and a non-leap year. * Test data: “2/29/2100” | False |
| 9 | February 29th shall only be valid on a leap year. | * Create an instance of Date with month 2, day 29, and a valid leap year. * Test data: “2/29/2000” | True |
| 10 | The valid range for the short non-February months shall be 1-30 | * Create an instance of Date with valid short month and year, but the day is greater than 30. * Test data: “4/31/2022” | False |
| 11 | The year shall not be negative | * Create an instance of Date with valid date and month, but the year is negative. * Test data: “1/12/-2012” | False |