**Phạm Trần Gia Hưng 2331200153**

# Practice Assignment 2

Save the source code, test cases and related test report as per question number. Make a zip folder of all the solutions and upload in Moodle.

## 1. **You need to develop a software based on Test Driven Development (TDD). The details of the software you need to implement as follows**.

The software will generate Personal Numbers. Let assume, the personal number consist of 10-digit number: YYMMDD-XYZC.

• The first six digits represent the date of birth with year, month and day, for example 640823, which is 23rd of August 1964.

• The following three digits, XYZ, is a serial number, where Z represents the person's gender. If the person is a female Z is an even number and odd if the person is male.

• The last digit, C, is a checksum and is calculated in the following manner:

◦ Multiply the digits in the date and serial number with 2, 1, 2, 1,...

For example:
6 4 0 8 2 3 – 3 2 3
2 1 2 1 2 1 2 1 2
12, 4, 0, 8, 4, 3, 6,2,6

◦ Add the resulting digits: 1+2+4+0+8+4+3+6+2+6=36

▪ If a multiplication result is larger than 10, it becomes the sum of the digits, for example 12→ 1+2

◦ If the summation [in this case it is 36] is greater than 10, take the last digit from the sum (the 6 in the above example) and subtract it from 10.
In the example it will be 10-6 = 4, the resulting number is the check sum C.

◦ If the result is 10 then the checksum C is 0.

◦ If the result is less than 10, then the checksum C will be the digit itself.

The Personal number class shall have the following methods:

• getDate() should return the date of birth, YYMMDD in some form.
• getYear() only return the year
• getMonth() only return the month
• getSex() return if the person is male of female (returning just Z is not allowed).
• getCheckSum() return the checksum digit
• When instantiating a personal number class, give the personal number to the constructor as a parameter.
• It shall not be possible to instantiate an invalid personal number. If the instantiation fails an exception shall be thrown.

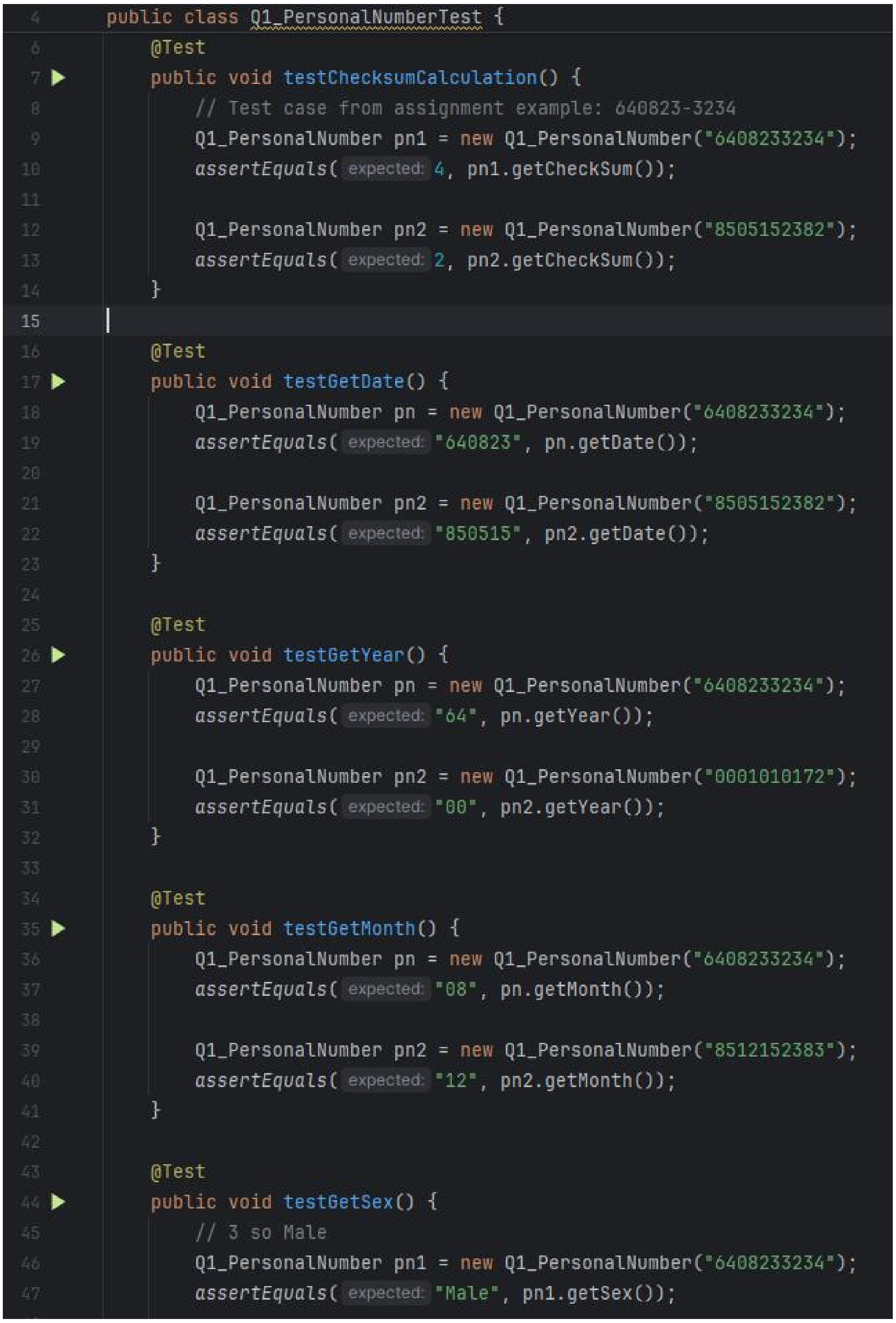
You are free to choose the types (integers, strings etc.) for your class by yourself.If needed, you can add more member functions, it is then a
At least you should have the following test cases for a class that represents the personal number:

1. A test case for checking the accuracy of checksum calculation
2. A test case for getDate() method
3. A test case for getYear() method
4. A test case for getMonth() method
5. A test case for getSex() method

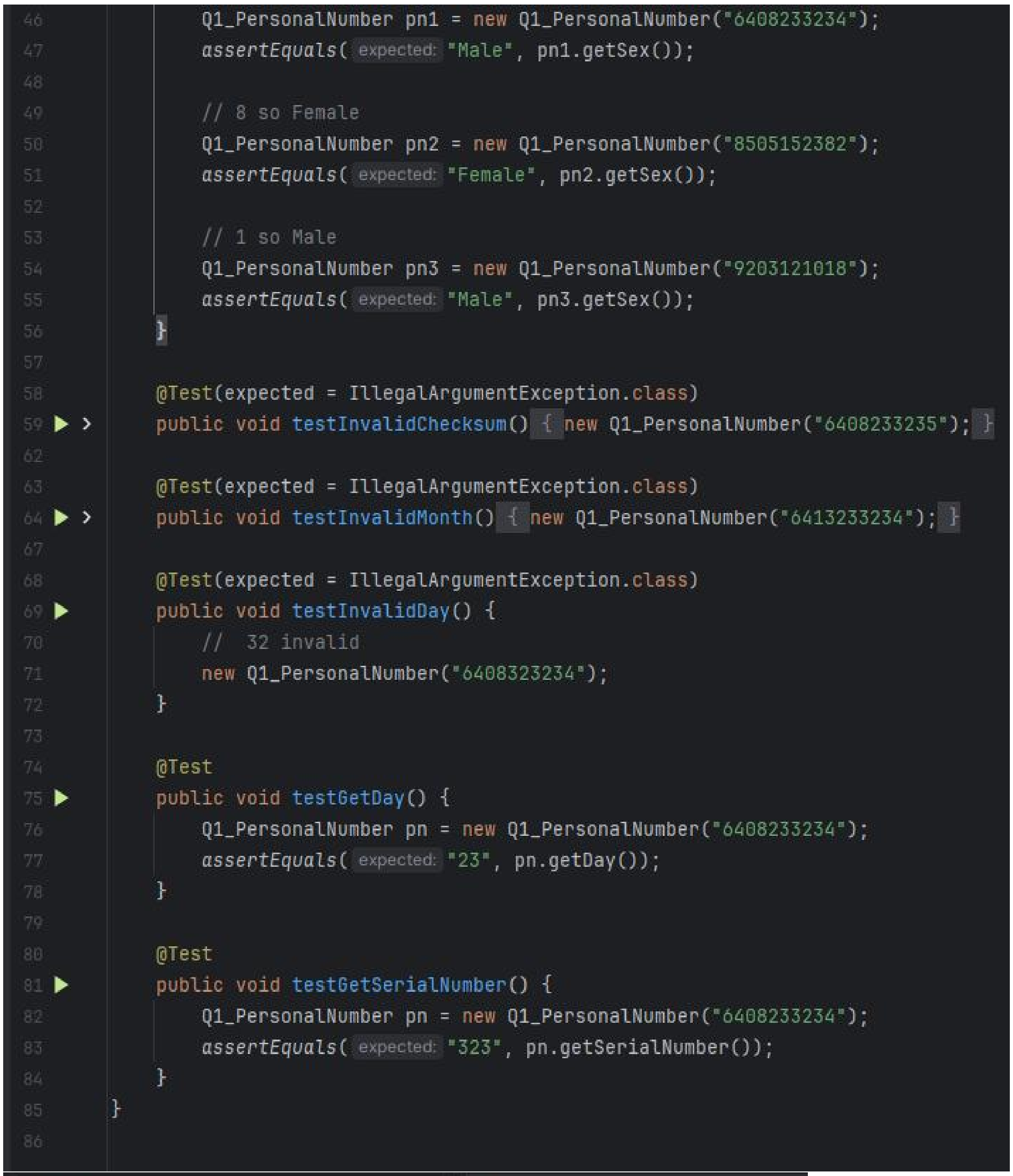
You should follow TDD. You should have the following steps in your development:

1. Add a test that fails
2. Make the code work
3. Run the test to see if it passes

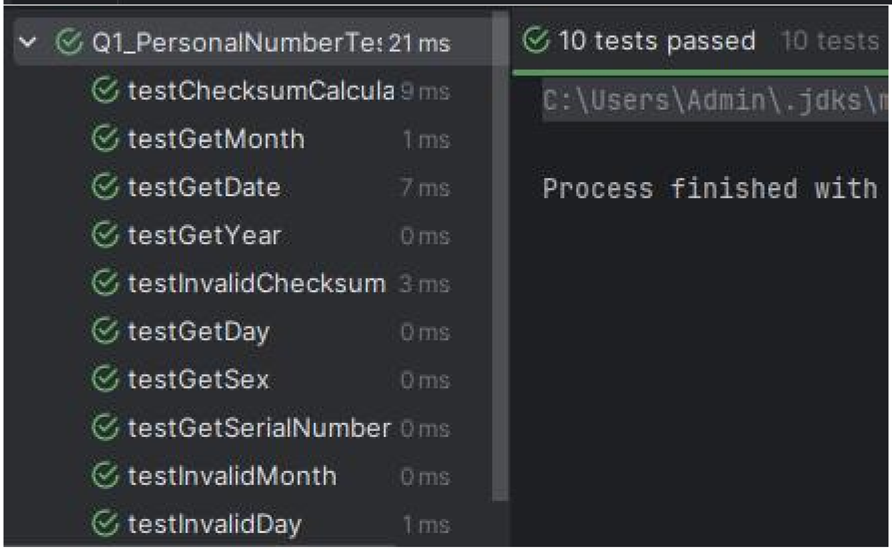
4. Modify and revise the code (Refactoring)
60 Points



```
public class Q1_PersonalNumberTest {
    @Test
    public void testChecksumCalculation() {
        // Test case from assignment example: 640823-3234
        Q1_PersonalNumber pn1 = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: 4, pn1.getCheckSum());

        Q1_PersonalNumber pn2 = new Q1_PersonalNumber("8505152382");
        assertEquals( expected: 2, pn2.getCheckSum());
    }

    @Test
    public void testGetDate() {
        Q1_PersonalNumber pn = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: "640823", pn.getDate());

        Q1_PersonalNumber pn2 = new Q1_PersonalNumber("8505152382");
        assertEquals( expected: "850515", pn2.getDate());
    }

    @Test
    public void testGetYear() {
        Q1_PersonalNumber pn = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: "64", pn.getYear());

        Q1_PersonalNumber pn2 = new Q1_PersonalNumber("0001010172");
        assertEquals( expected: "00", pn2.getYear());
    }

    @Test
    public void testGetMonth() {
        Q1_PersonalNumber pn = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: "08", pn.getMonth());

        Q1_PersonalNumber pn2 = new Q1_PersonalNumber("8512152383");
        assertEquals( expected: "12", pn2.getMonth());
    }

    @Test
    public void testGetSex() {
        // 3 so Male
        Q1_PersonalNumber pn1 = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: "Male", pn1.getSex());
```



```java
        Q1_PersonalNumber pn1 = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: "Male", pn1.getSex());

        // 8 so Female
        Q1_PersonalNumber pn2 = new Q1_PersonalNumber("8505152382");
        assertEquals( expected: "Female", pn2.getSex());

        // 1 so Male
        Q1_PersonalNumber pn3 = new Q1_PersonalNumber("9203121018");
        assertEquals( expected: "Male", pn3.getSex());
    }

    @Test(expected = IllegalArgumentException.class)
    public void testInvalidChecksum() { new Q1_PersonalNumber("6408233235"); }

    @Test(expected = IllegalArgumentException.class)
    public void testInvalidMonth() { new Q1_PersonalNumber("6413233234"); }

    @Test(expected = IllegalArgumentException.class)
    public void testInvalidDay() {
        //  32 invalid
        new Q1_PersonalNumber("6408323234");
    }

    @Test
    public void testGetDay() {
        Q1_PersonalNumber pn = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: "23", pn.getDay());
    }

    @Test
    public void testGetSerialNumber() {
        Q1_PersonalNumber pn = new Q1_PersonalNumber("6408233234");
        assertEquals( expected: "323", pn.getSerialNumber());
    }
}
```

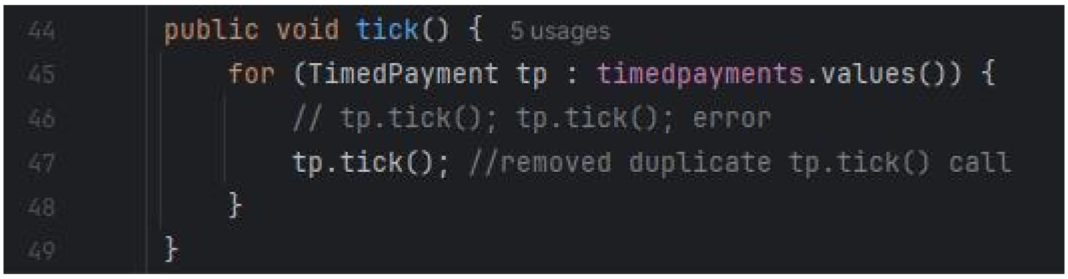


```
✓ Q1_PersonalNumberTe: 21 ms        ✓ 10 tests passed  10 tests
    ✓ testChecksumCalcula 9 ms
    ✓ testGetMonth        1 ms      C:\Users\Admin\.jdks\
    ✓ testGetDate         7 ms
    ✓ testGetYear         0 ms      Process finished with
    ✓ testInvalidChecksum 3 ms
    ✓ testGetDay          0 ms
    ✓ testGetSex          0 ms
    ✓ testGetSerialNumber 0 ms
    ✓ testInvalidMonth    0 ms
    ✓ testInvalidDay      1 ms
```

**2. You are given a template for the Currency and Money classes. For these two classes, at first, write test** cases for the methods of each class, then you have to add code to the methods of these classes (You need to follow Test Driven Development –TDD).The Bank and Account classes were written by a bad programmer. When you are confident that your Money and Currency classes work as intended, write test cases for the Bank and Account classes and find the bugs.
30 Points

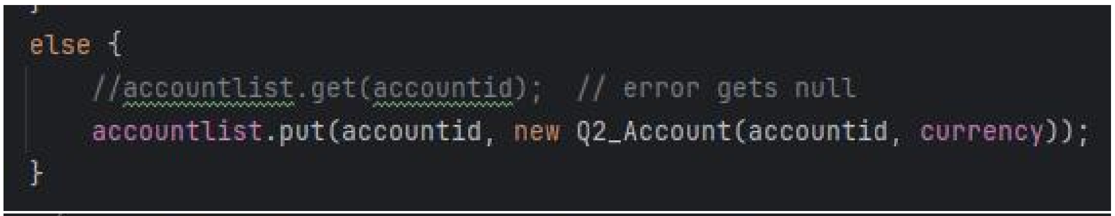
**Bug:**

**Account:**

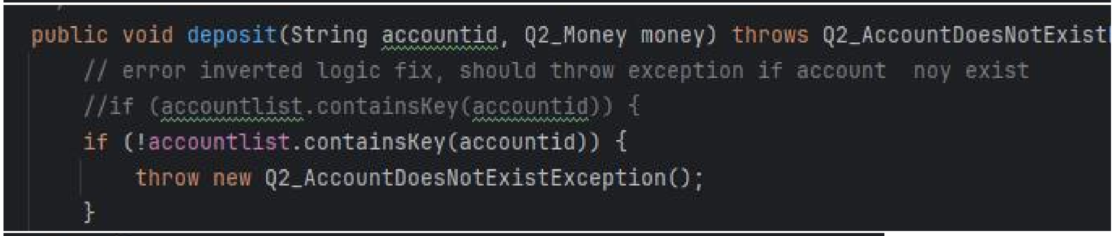


```
public void tick() {  5 usages
    for (TimedPayment tp : timedpayments.values()) {
        // tp.tick(); tp.tick(); error
        tp.tick(); //removed duplicate tp.tick() call
    }
}
```

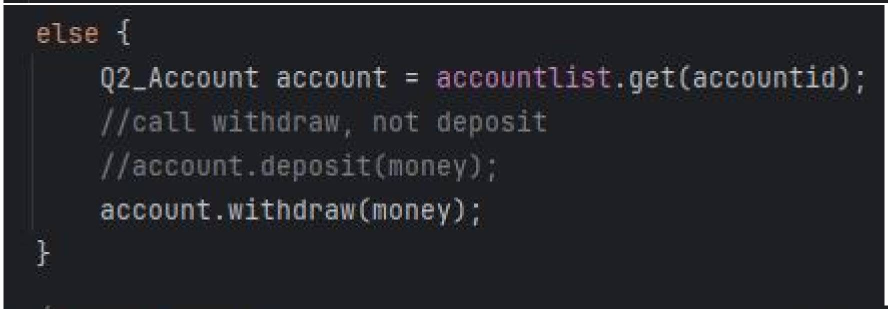
**Bank:**



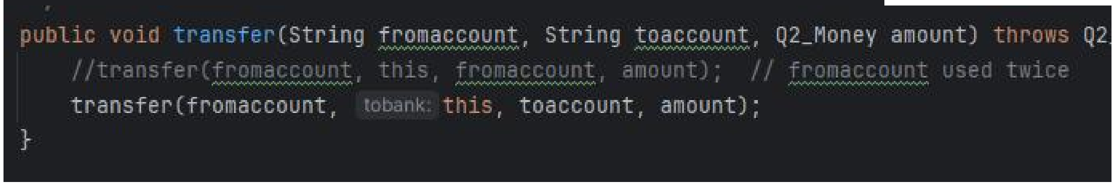
```
else {
    //accountlist.get(accountid);  // error gets null
    accountlist.put(accountid, new Q2_Account(accountid, currency));
}
```



```
public void deposit(String accountid, Q2_Money money) throws Q2_AccountDoesNotExist
    // error inverted logic fix, should throw exception if account  noy exist
    //if (accountlist.containsKey(accountid)) {
    if (!accountlist.containsKey(accountid)) {
        throw new Q2_AccountDoesNotExistException();
    }
```

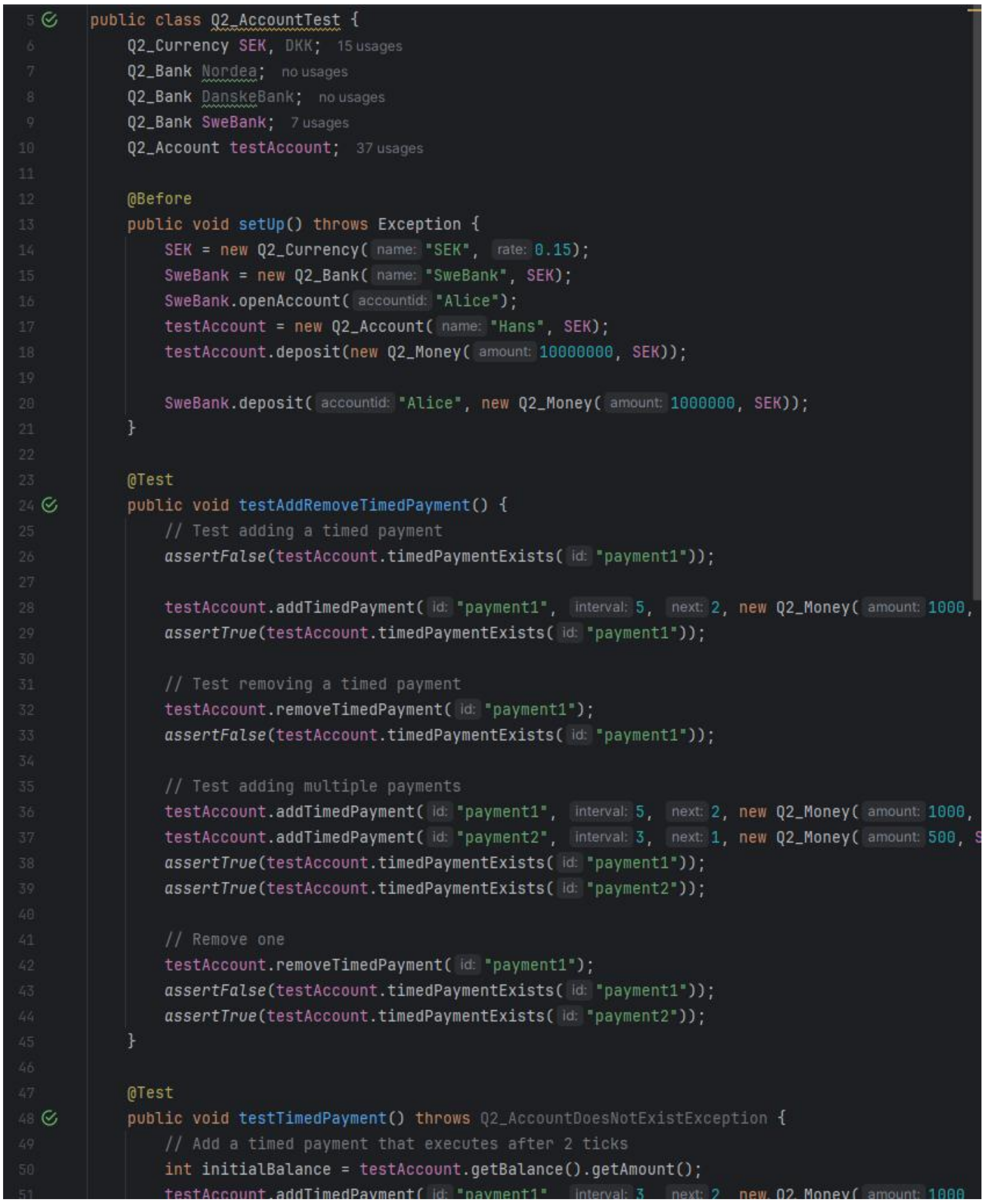


```
else {
    Q2_Account account = accountlist.get(accountid);
    //call withdraw, not deposit
    //account.deposit(money);
    account.withdraw(money);
}
```

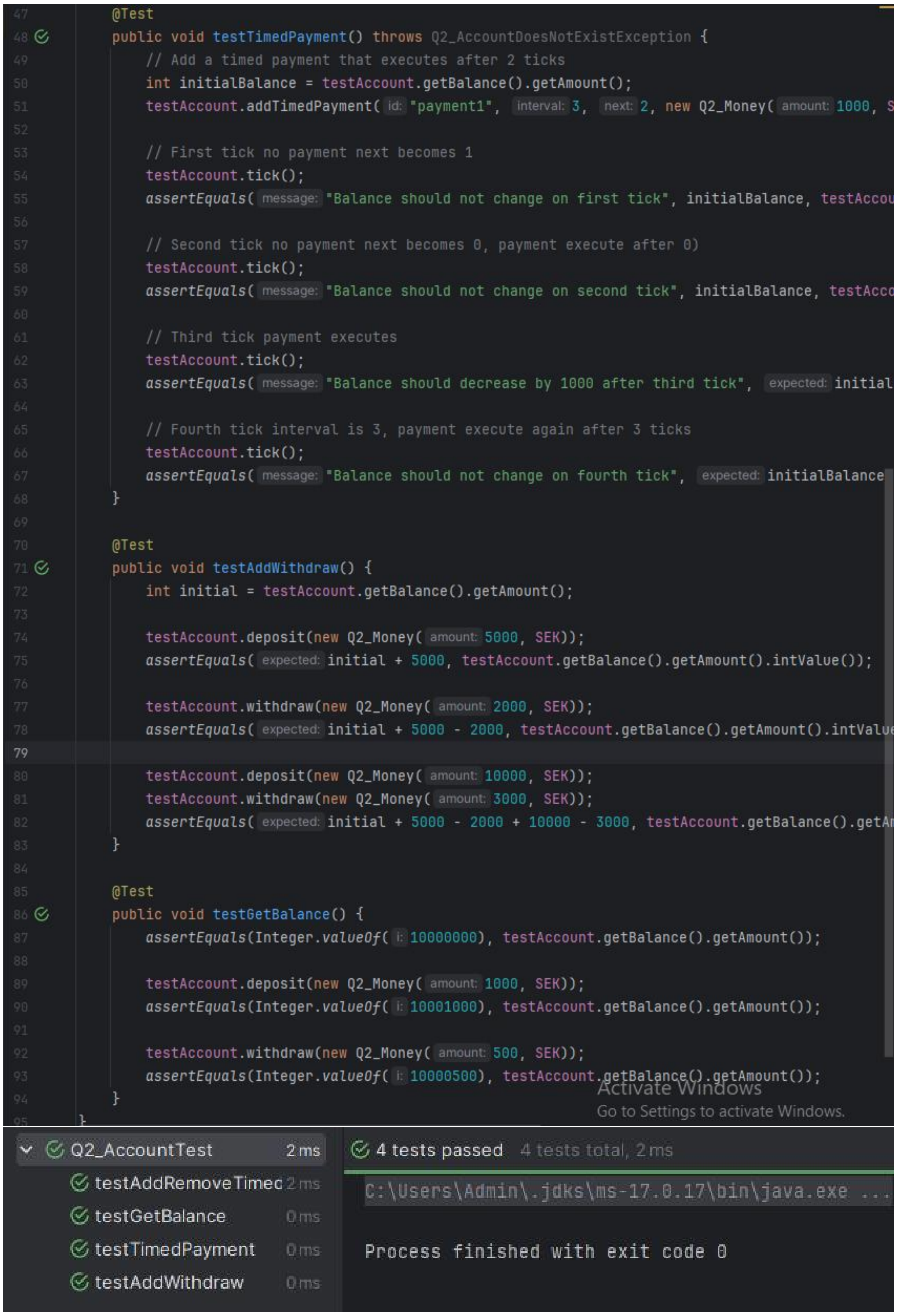


```
public void transfer(String fromaccount, String toaccount, Q2_Money amount) throws Q2
    //transfer(fromaccount, this, fromaccount, amount);  // fromaccount used twice
    transfer(fromaccount, tobank: this, toaccount, amount);
}
```

**- Account test:**



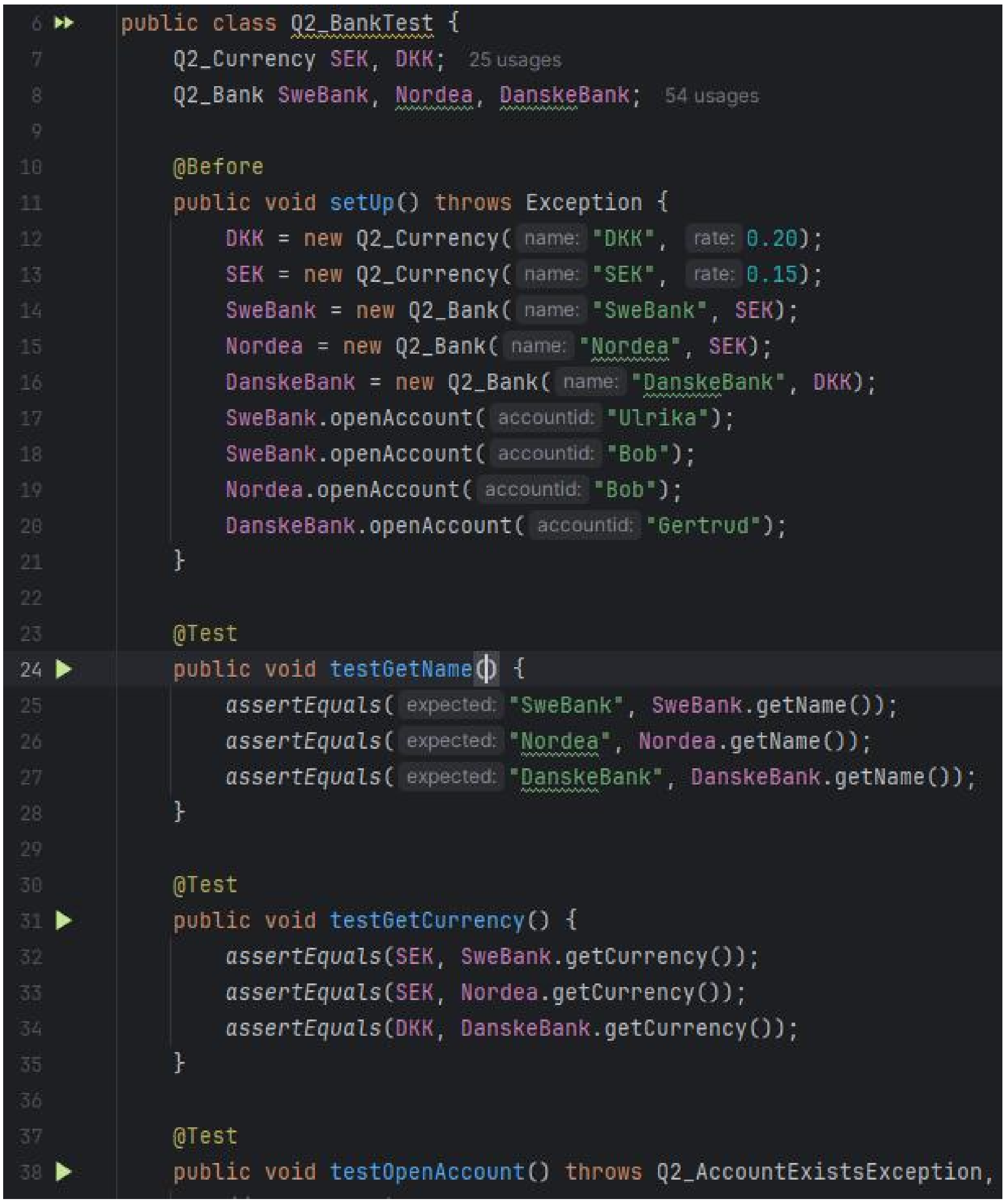
```
public class Q2_AccountTest {
    Q2_Currency SEK, DKK;  15 usages
    Q2_Bank Nordea;  no usages
    Q2_Bank DanskeBank;  no usages
    Q2_Bank SweBank;  7 usages
    Q2_Account testAccount;  37 usages

    @Before
    public void setUp() throws Exception {
        SEK = new Q2_Currency( name: "SEK",  rate: 0.15);
        SweBank = new Q2_Bank( name: "SweBank", SEK);
        SweBank.openAccount( accountid: "Alice");
        testAccount = new Q2_Account( name: "Hans", SEK);
        testAccount.deposit(new Q2_Money( amount: 10000000, SEK));

        SweBank.deposit( accountid: "Alice", new Q2_Money( amount: 1000000, SEK));
    }

    @Test
    public void testAddRemoveTimedPayment() {
        // Test adding a timed payment
        assertFalse(testAccount.timedPaymentExists( id: "payment1"));

        testAccount.addTimedPayment( id: "payment1",  interval: 5,  next: 2, new Q2_Money( amount: 1000,
        assertTrue(testAccount.timedPaymentExists( id: "payment1"));

        // Test removing a timed payment
        testAccount.removeTimedPayment( id: "payment1");
        assertFalse(testAccount.timedPaymentExists( id: "payment1"));

        // Test adding multiple payments
        testAccount.addTimedPayment( id: "payment1",  interval: 5,  next: 2, new Q2_Money( amount: 1000,
        testAccount.addTimedPayment( id: "payment2",  interval: 3,  next: 1, new Q2_Money( amount: 500, S
        assertTrue(testAccount.timedPaymentExists( id: "payment1"));
        assertTrue(testAccount.timedPaymentExists( id: "payment2"));

        // Remove one
        testAccount.removeTimedPayment( id: "payment1");
        assertFalse(testAccount.timedPaymentExists( id: "payment1"));
        assertTrue(testAccount.timedPaymentExists( id: "payment2"));
    }

    @Test
    public void testTimedPayment() throws Q2_AccountDoesNotExistException {
        // Add a timed payment that executes after 2 ticks
        int initialBalance = testAccount.getBalance().getAmount();
        testAccount.addTimedPayment( id: "payment1"  interval: 3  next: 2  new Q2_Money( amount: 1000
```



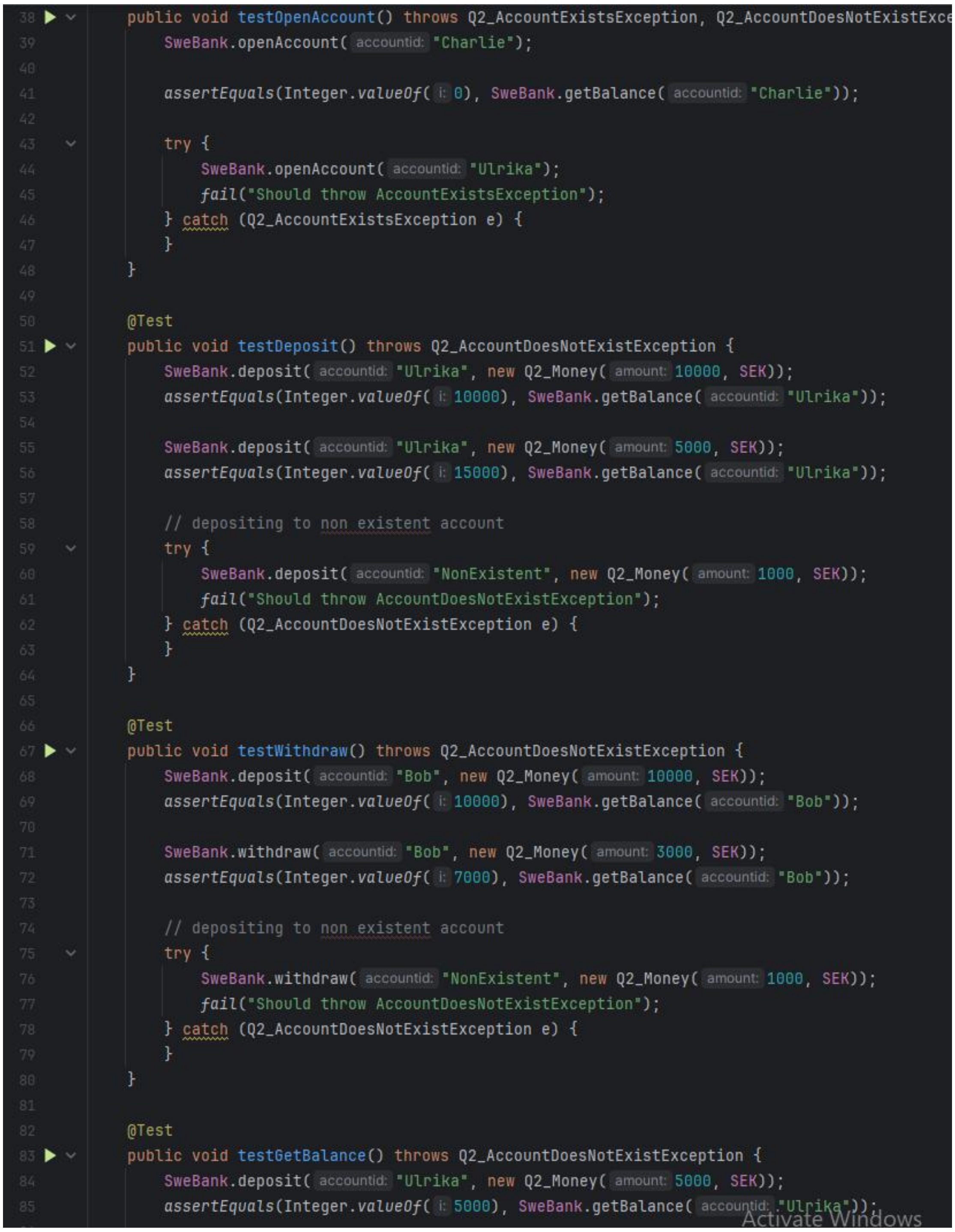
```java
    @Test
    public void testTimedPayment() throws Q2_AccountDoesNotExistException {
        // Add a timed payment that executes after 2 ticks
        int initialBalance = testAccount.getBalance().getAmount();
        testAccount.addTimedPayment( id: "payment1",  interval: 3,  next: 2, new Q2_Money( amount: 1000, S

        // First tick no payment next becomes 1
        testAccount.tick();
        assertEquals( message: "Balance should not change on first tick", initialBalance, testAccou

        // Second tick no payment next becomes 0, payment execute after 0)
        testAccount.tick();
        assertEquals( message: "Balance should not change on second tick", initialBalance, testAcco

        // Third tick payment executes
        testAccount.tick();
        assertEquals( message: "Balance should decrease by 1000 after third tick",  expected: initial

        // Fourth tick interval is 3, payment execute again after 3 ticks
        testAccount.tick();
        assertEquals( message: "Balance should not change on fourth tick",  expected: initialBalance
    }

    @Test
    public void testAddWithdraw() {
        int initial = testAccount.getBalance().getAmount();

        testAccount.deposit(new Q2_Money( amount: 5000, SEK));
        assertEquals( expected: initial + 5000, testAccount.getBalance().getAmount().intValue());

        testAccount.withdraw(new Q2_Money( amount: 2000, SEK));
        assertEquals( expected: initial + 5000 - 2000, testAccount.getBalance().getAmount().intValue

        testAccount.deposit(new Q2_Money( amount: 10000, SEK));
        testAccount.withdraw(new Q2_Money( amount: 3000, SEK));
        assertEquals( expected: initial + 5000 - 2000 + 10000 - 3000, testAccount.getBalance().getAm
    }

    @Test
    public void testGetBalance() {
        assertEquals(Integer.valueOf( i: 10000000), testAccount.getBalance().getAmount());

        testAccount.deposit(new Q2_Money( amount: 1000, SEK));
        assertEquals(Integer.valueOf( i: 10001000), testAccount.getBalance().getAmount());

        testAccount.withdraw(new Q2_Money( amount: 500, SEK));
        assertEquals(Integer.valueOf( i: 10000500), testAccount.getBalance().getAmount());
    }
}
```

```
Q2_AccountTest            2 ms     4 tests passed  4 tests total, 2 ms
  testAddRemoveTimed      2 ms     C:\Users\Admin\.jdks\ms-17.0.17\bin\java.exe ...
  testGetBalance          0 ms
  testTimedPayment        0 ms     Process finished with exit code 0
  testAddWithdraw         0 ms
```

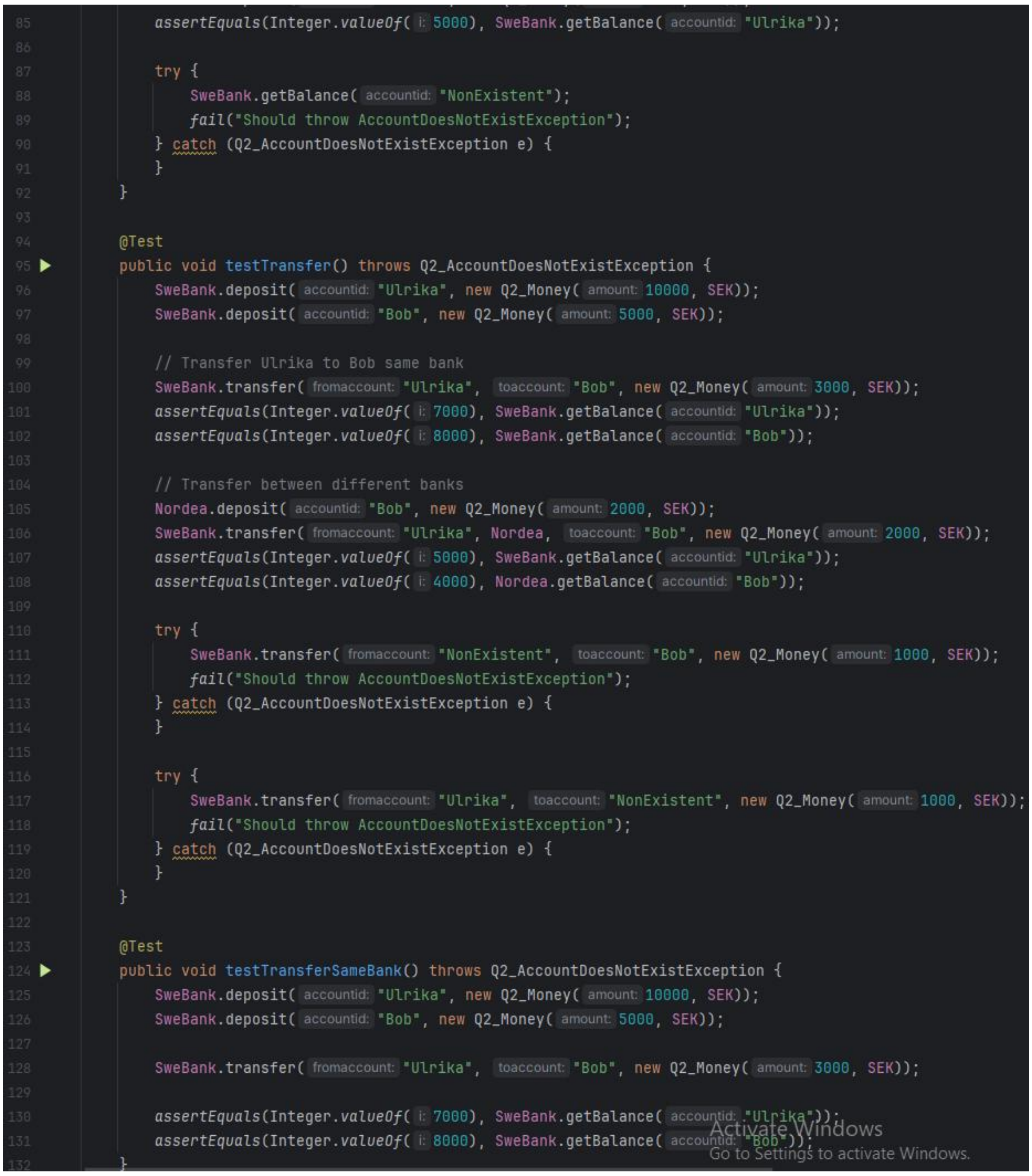
**- Bank test:**



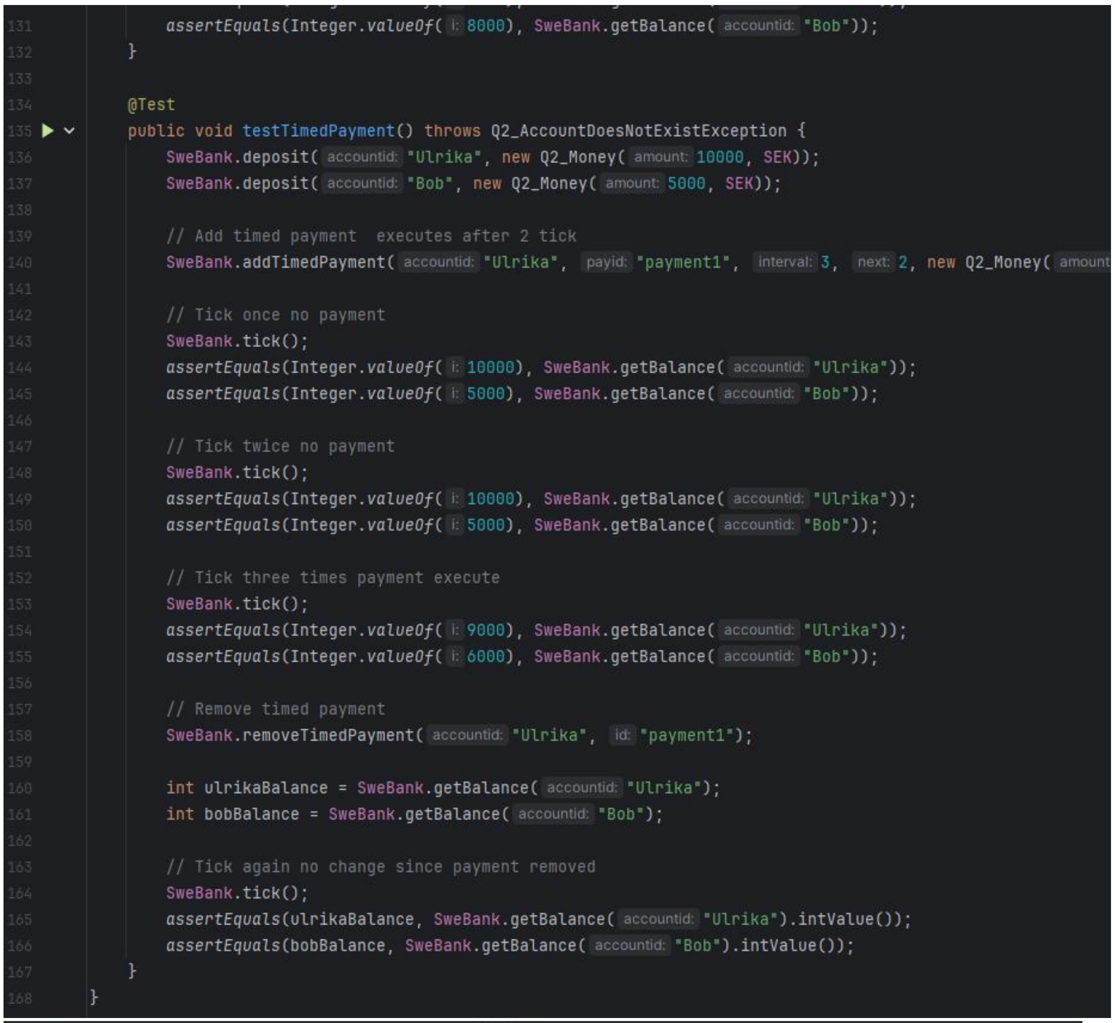
```java
public class Q2_BankTest {
    Q2_Currency SEK, DKK;  25 usages
    Q2_Bank SweBank, Nordea, DanskeBank;  54 usages

    @Before
    public void setUp() throws Exception {
        DKK = new Q2_Currency( name: "DKK",  rate: 0.20);
        SEK = new Q2_Currency( name: "SEK",  rate: 0.15);
        SweBank = new Q2_Bank( name: "SweBank", SEK);
        Nordea = new Q2_Bank( name: "Nordea", SEK);
        DanskeBank = new Q2_Bank( name: "DanskeBank", DKK);
        SweBank.openAccount( accountid: "Ulrika");
        SweBank.openAccount( accountid: "Bob");
        Nordea.openAccount( accountid: "Bob");
        DanskeBank.openAccount( accountid: "Gertrud");
    }

    @Test
    public void testGetName() {
        assertEquals( expected: "SweBank", SweBank.getName());
        assertEquals( expected: "Nordea", Nordea.getName());
        assertEquals( expected: "DanskeBank", DanskeBank.getName());
    }

    @Test
    public void testGetCurrency() {
        assertEquals(SEK, SweBank.getCurrency());
        assertEquals(SEK, Nordea.getCurrency());
        assertEquals(DKK, DanskeBank.getCurrency());
    }

    @Test
    public void testOpenAccount() throws Q2_AccountExistsException,
```



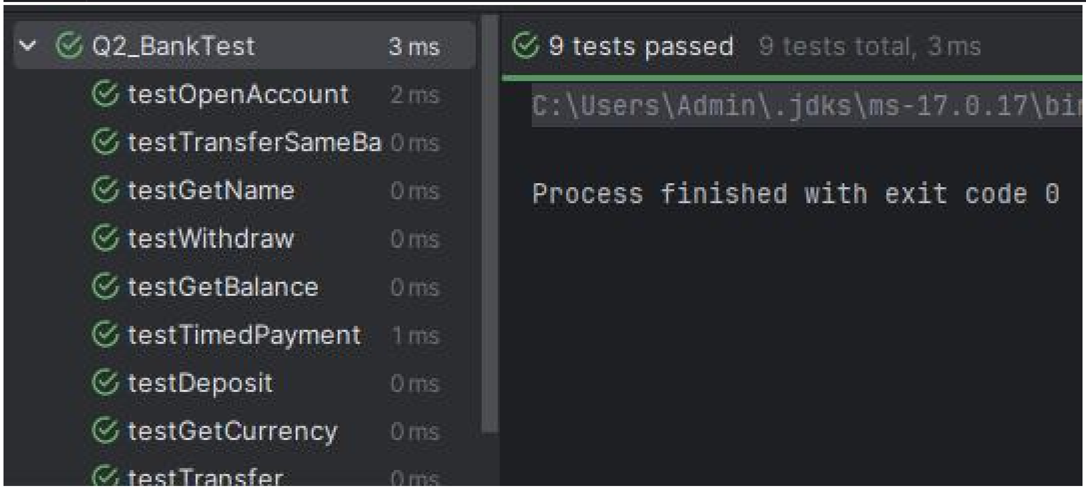
```
    public void testOpenAccount() throws Q2_AccountExistsException, Q2_AccountDoesNotExistExce
        SweBank.openAccount( accountId: "Charlie");

        assertEquals(Integer.valueOf( i: 0), SweBank.getBalance( accountId: "Charlie"));

        try {
            SweBank.openAccount( accountId: "Ulrika");
            fail("Should throw AccountExistsException");
        } catch (Q2_AccountExistsException e) {
        }
    }

    @Test
    public void testDeposit() throws Q2_AccountDoesNotExistException {
        SweBank.deposit( accountId: "Ulrika", new Q2_Money( amount: 10000, SEK));
        assertEquals(Integer.valueOf( i: 10000), SweBank.getBalance( accountId: "Ulrika"));

        SweBank.deposit( accountId: "Ulrika", new Q2_Money( amount: 5000, SEK));
        assertEquals(Integer.valueOf( i: 15000), SweBank.getBalance( accountId: "Ulrika"));

        // depositing to non existent account
        try {
            SweBank.deposit( accountId: "NonExistent", new Q2_Money( amount: 1000, SEK));
            fail("Should throw AccountDoesNotExistException");
        } catch (Q2_AccountDoesNotExistException e) {
        }
    }

    @Test
    public void testWithdraw() throws Q2_AccountDoesNotExistException {
        SweBank.deposit( accountId: "Bob", new Q2_Money( amount: 10000, SEK));
        assertEquals(Integer.valueOf( i: 10000), SweBank.getBalance( accountId: "Bob"));

        SweBank.withdraw( accountId: "Bob", new Q2_Money( amount: 3000, SEK));
        assertEquals(Integer.valueOf( i: 7000), SweBank.getBalance( accountId: "Bob"));

        // depositing to non existent account
        try {
            SweBank.withdraw( accountId: "NonExistent", new Q2_Money( amount: 1000, SEK));
            fail("Should throw AccountDoesNotExistException");
        } catch (Q2_AccountDoesNotExistException e) {
        }
    }

    @Test
    public void testGetBalance() throws Q2_AccountDoesNotExistException {
        SweBank.deposit( accountId: "Ulrika", new Q2_Money( amount: 5000, SEK));
        assertEquals(Integer.valueOf( i: 5000), SweBank.getBalance( accountId: "Ulrika"));
```



```java
        assertEquals(Integer.valueOf( i: 5000), SweBank.getBalance( accountid: "Ulrika"));

        try {
            SweBank.getBalance( accountid: "NonExistent");
            fail("Should throw AccountDoesNotExistException");
        } catch (Q2_AccountDoesNotExistException e) {
        }
    }

    @Test
    public void testTransfer() throws Q2_AccountDoesNotExistException {
        SweBank.deposit( accountid: "Ulrika", new Q2_Money( amount: 10000, SEK));
        SweBank.deposit( accountid: "Bob", new Q2_Money( amount: 5000, SEK));

        // Transfer Ulrika to Bob same bank
        SweBank.transfer( fromaccount: "Ulrika",  toaccount: "Bob", new Q2_Money( amount: 3000, SEK));
        assertEquals(Integer.valueOf( i: 7000), SweBank.getBalance( accountid: "Ulrika"));
        assertEquals(Integer.valueOf( i: 8000), SweBank.getBalance( accountid: "Bob"));

        // Transfer between different banks
        Nordea.deposit( accountid: "Bob", new Q2_Money( amount: 2000, SEK));
        SweBank.transfer( fromaccount: "Ulrika", Nordea,  toaccount: "Bob", new Q2_Money( amount: 2000, SEK));
        assertEquals(Integer.valueOf( i: 5000), SweBank.getBalance( accountid: "Ulrika"));
        assertEquals(Integer.valueOf( i: 4000), Nordea.getBalance( accountid: "Bob"));

        try {
            SweBank.transfer( fromaccount: "NonExistent",  toaccount: "Bob", new Q2_Money( amount: 1000, SEK));
            fail("Should throw AccountDoesNotExistException");
        } catch (Q2_AccountDoesNotExistException e) {
        }

        try {
            SweBank.transfer( fromaccount: "Ulrika",  toaccount: "NonExistent", new Q2_Money( amount: 1000, SEK));
            fail("Should throw AccountDoesNotExistException");
        } catch (Q2_AccountDoesNotExistException e) {
        }
    }

    @Test
    public void testTransferSameBank() throws Q2_AccountDoesNotExistException {
        SweBank.deposit( accountid: "Ulrika", new Q2_Money( amount: 10000, SEK));
        SweBank.deposit( accountid: "Bob", new Q2_Money( amount: 5000, SEK));

        SweBank.transfer( fromaccount: "Ulrika",  toaccount: "Bob", new Q2_Money( amount: 3000, SEK));

        assertEquals(Integer.valueOf( i: 7000), SweBank.getBalance( accountid: "Ulrika"));
        assertEquals(Integer.valueOf( i: 8000), SweBank.getBalance( accountid: "Bob"));
    }
```



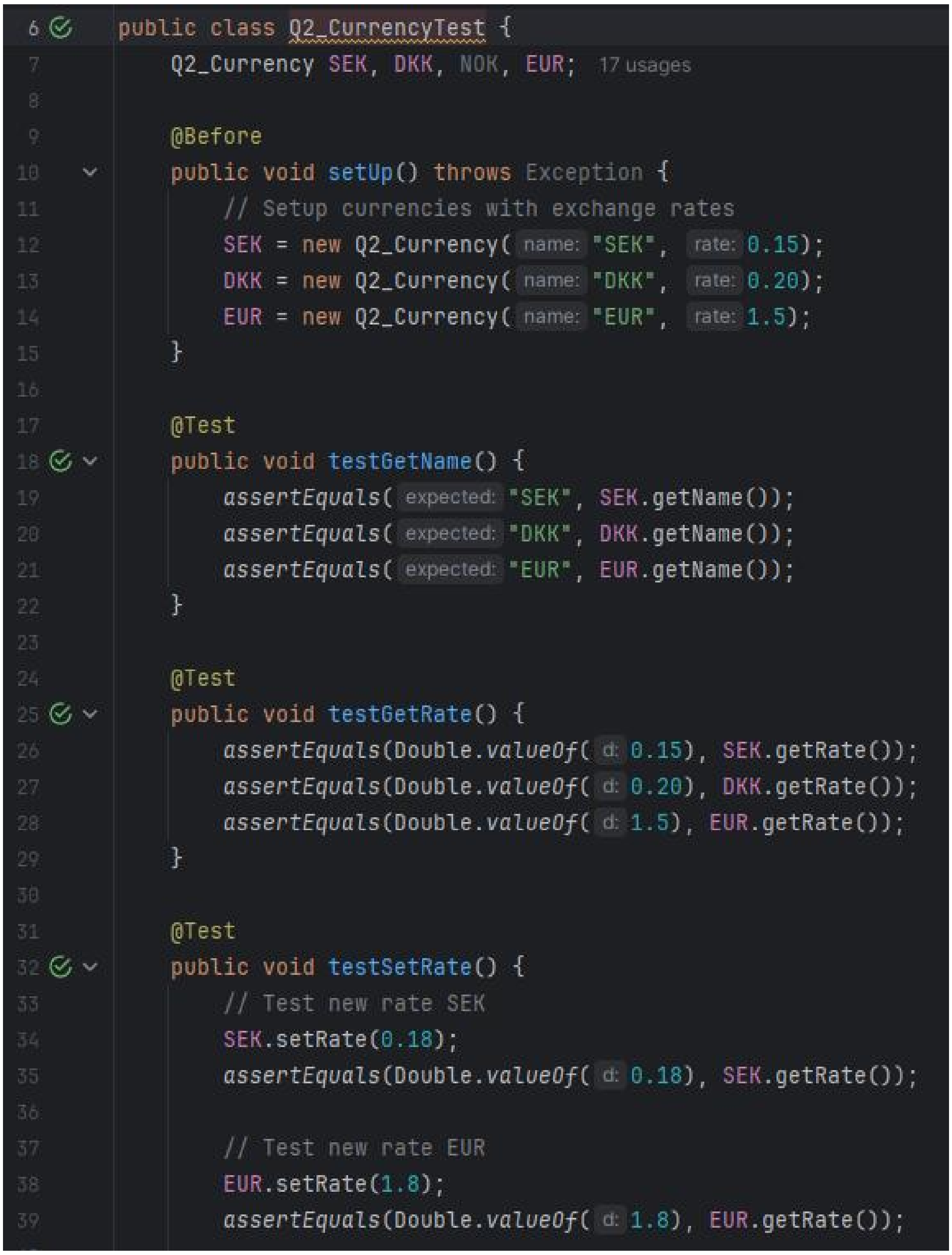
```
        assertEquals(Integer.valueOf( i: 8000), SweBank.getBalance( accountid: "Bob"));
    }

    @Test
    public void testTimedPayment() throws Q2_AccountDoesNotExistException {
        SweBank.deposit( accountid: "Ulrika", new Q2_Money( amount: 10000, SEK));
        SweBank.deposit( accountid: "Bob", new Q2_Money( amount: 5000, SEK));

        // Add timed payment  executes after 2 tick
        SweBank.addTimedPayment( accountid: "Ulrika",  payid: "payment1",  interval: 3,  next: 2,  new Q2_Money( amount

        // Tick once no payment
        SweBank.tick();
        assertEquals(Integer.valueOf( i: 10000), SweBank.getBalance( accountid: "Ulrika"));
        assertEquals(Integer.valueOf( i: 5000), SweBank.getBalance( accountid: "Bob"));

        // Tick twice no payment
        SweBank.tick();
        assertEquals(Integer.valueOf( i: 10000), SweBank.getBalance( accountid: "Ulrika"));
        assertEquals(Integer.valueOf( i: 5000), SweBank.getBalance( accountid: "Bob"));

        // Tick three times payment execute
        SweBank.tick();
        assertEquals(Integer.valueOf( i: 9000), SweBank.getBalance( accountid: "Ulrika"));
        assertEquals(Integer.valueOf( i: 6000), SweBank.getBalance( accountid: "Bob"));

        // Remove timed payment
        SweBank.removeTimedPayment( accountid: "Ulrika",  id: "payment1");

        int ulrikaBalance = SweBank.getBalance( accountid: "Ulrika");
        int bobBalance = SweBank.getBalance( accountid: "Bob");

        // Tick again no change since payment removed
        SweBank.tick();
        assertEquals(ulrikaBalance, SweBank.getBalance( accountid: "Ulrika").intValue());
        assertEquals(bobBalance, SweBank.getBalance( accountid: "Bob").intValue());
    }
}
```



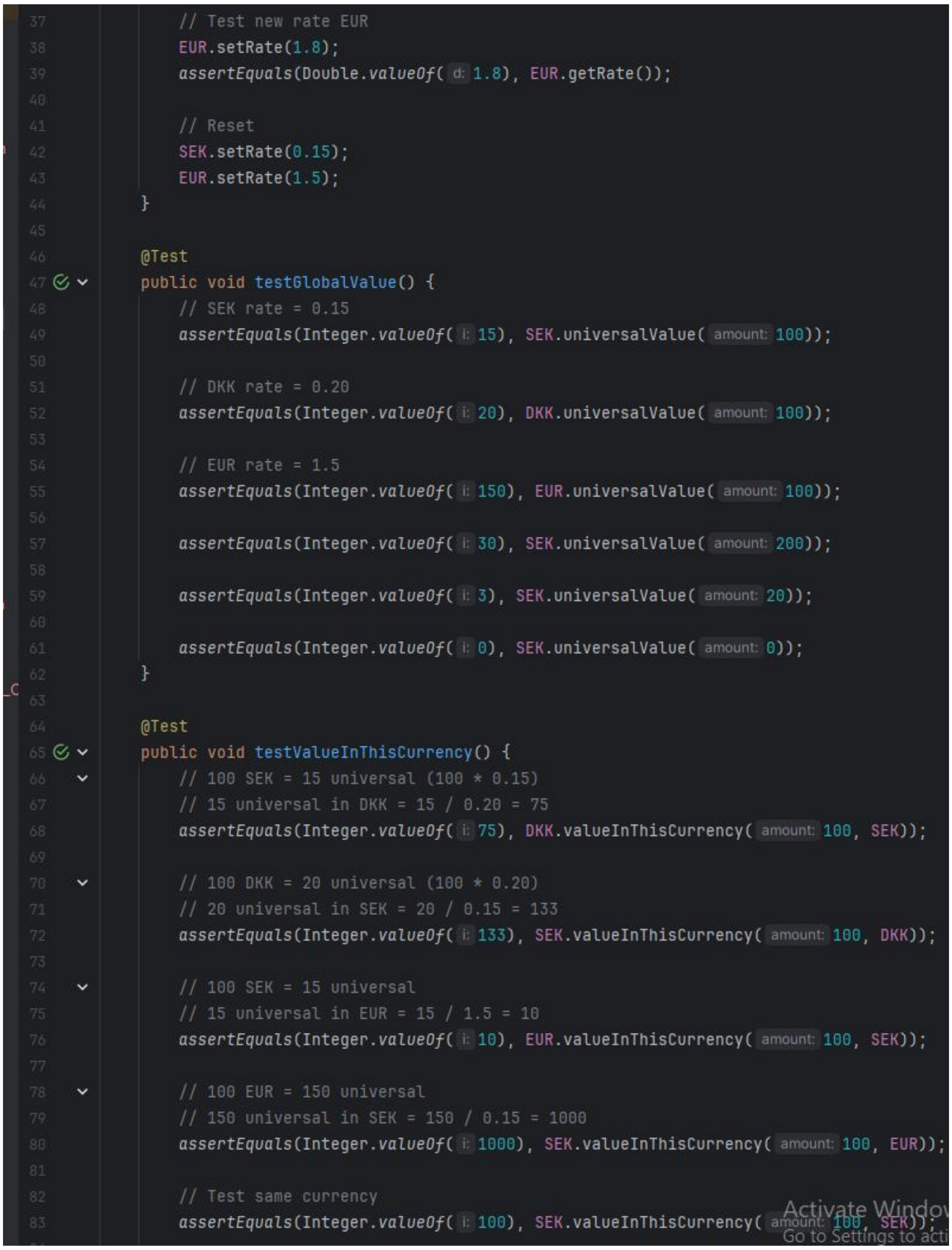
```
Q2_BankTest              3 ms
    testOpenAccount      2 ms
    testTransferSameBa   0 ms
    testGetName          0 ms
    testWithdraw         0 ms
    testGetBalance       0 ms
    testTimedPayment     1 ms
    testDeposit          0 ms
    testGetCurrency      0 ms
    testTransfer         0 ms
```

```
9 tests passed   9 tests total, 3 ms

C:\Users\Admin\.jdks\ms-17.0.17\bi

Process finished with exit code 0
```

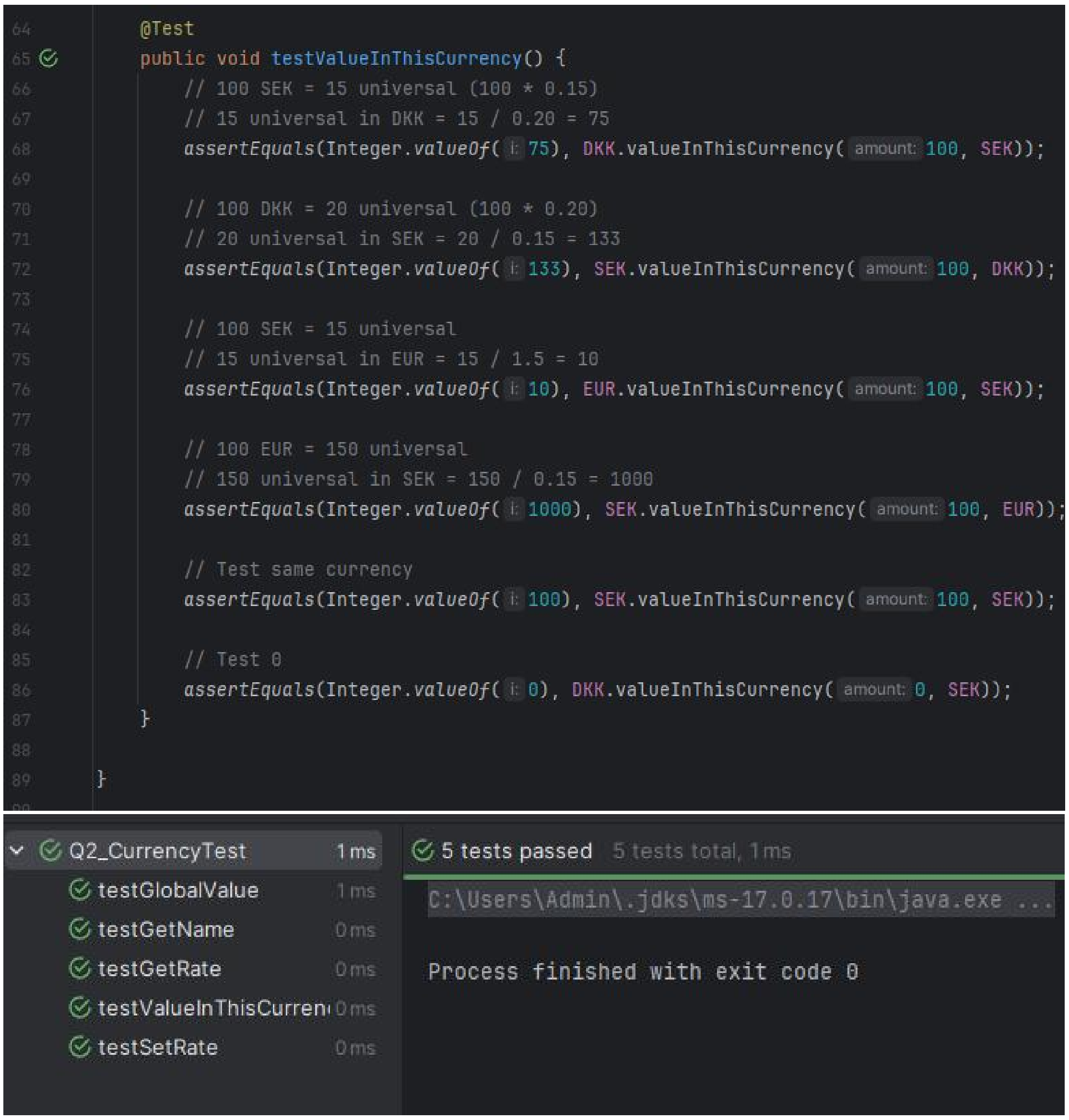
## -Currency test



```
public class Q2_CurrencyTest {
    Q2_Currency SEK, DKK, NOK, EUR;  17 usages

    @Before
    public void setUp() throws Exception {
        // Setup currencies with exchange rates
        SEK = new Q2_Currency( name: "SEK",  rate: 0.15);
        DKK = new Q2_Currency( name: "DKK",  rate: 0.20);
        EUR = new Q2_Currency( name: "EUR",  rate: 1.5);
    }

    @Test
    public void testGetName() {
        assertEquals( expected: "SEK", SEK.getName());
        assertEquals( expected: "DKK", DKK.getName());
        assertEquals( expected: "EUR", EUR.getName());
    }

    @Test
    public void testGetRate() {
        assertEquals(Double.valueOf( d: 0.15), SEK.getRate());
        assertEquals(Double.valueOf( d: 0.20), DKK.getRate());
        assertEquals(Double.valueOf( d: 1.5), EUR.getRate());
    }

    @Test
    public void testSetRate() {
        // Test new rate SEK
        SEK.setRate(0.18);
        assertEquals(Double.valueOf( d: 0.18), SEK.getRate());

        // Test new rate EUR
        EUR.setRate(1.8);
        assertEquals(Double.valueOf( d: 1.8), EUR.getRate());
```



```java
        // Test new rate EUR
        EUR.setRate(1.8);
        assertEquals(Double.valueOf( d: 1.8), EUR.getRate());

        // Reset
        SEK.setRate(0.15);
        EUR.setRate(1.5);
    }

    @Test
    public void testGlobalValue() {
        // SEK rate = 0.15
        assertEquals(Integer.valueOf( i: 15), SEK.universalValue( amount: 100));

        // DKK rate = 0.20
        assertEquals(Integer.valueOf( i: 20), DKK.universalValue( amount: 100));

        // EUR rate = 1.5
        assertEquals(Integer.valueOf( i: 150), EUR.universalValue( amount: 100));

        assertEquals(Integer.valueOf( i: 30), SEK.universalValue( amount: 200));

        assertEquals(Integer.valueOf( i: 3), SEK.universalValue( amount: 20));

        assertEquals(Integer.valueOf( i: 0), SEK.universalValue( amount: 0));
    }

    @Test
    public void testValueInThisCurrency() {
        // 100 SEK = 15 universal (100 * 0.15)
        // 15 universal in DKK = 15 / 0.20 = 75
        assertEquals(Integer.valueOf( i: 75), DKK.valueInThisCurrency( amount: 100, SEK));

        // 100 DKK = 20 universal (100 * 0.20)
        // 20 universal in SEK = 20 / 0.15 = 133
        assertEquals(Integer.valueOf( i: 133), SEK.valueInThisCurrency( amount: 100, DKK));

        // 100 SEK = 15 universal
        // 15 universal in EUR = 15 / 1.5 = 10
        assertEquals(Integer.valueOf( i: 10), EUR.valueInThisCurrency( amount: 100, SEK));

        // 100 EUR = 150 universal
        // 150 universal in SEK = 150 / 0.15 = 1000
        assertEquals(Integer.valueOf( i: 1000), SEK.valueInThisCurrency( amount: 100, EUR));

        // Test same currency
        assertEquals(Integer.valueOf( i: 100), SEK.valueInThisCurrency( amount: 100, SEK));
```

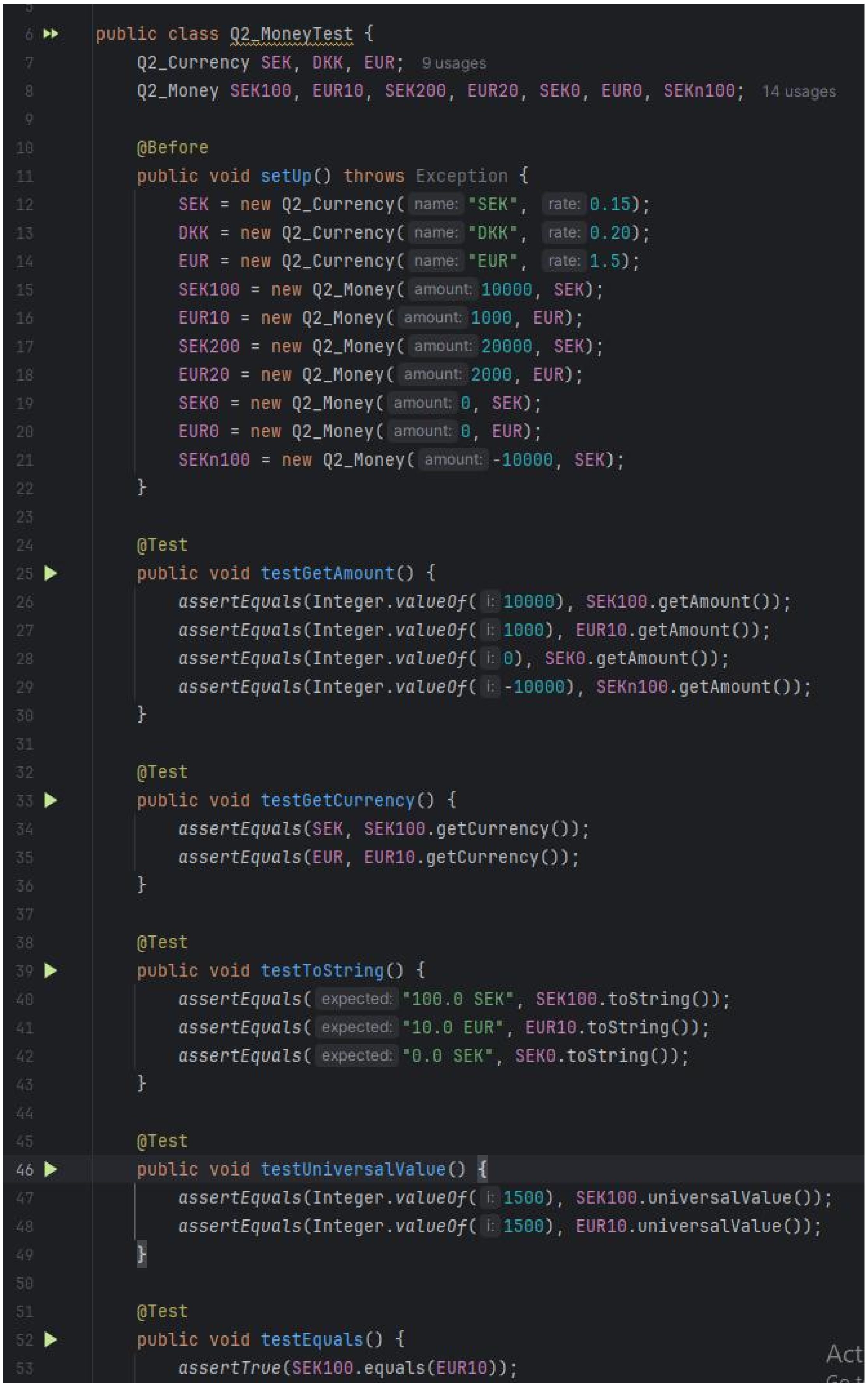


```java
    @Test
    public void testValueInThisCurrency() {
        // 100 SEK = 15 universal (100 * 0.15)
        // 15 universal in DKK = 15 / 0.20 = 75
        assertEquals(Integer.valueOf( i: 75), DKK.valueInThisCurrency( amount: 100, SEK));

        // 100 DKK = 20 universal (100 * 0.20)
        // 20 universal in SEK = 20 / 0.15 = 133
        assertEquals(Integer.valueOf( i: 133), SEK.valueInThisCurrency( amount: 100, DKK));

        // 100 SEK = 15 universal
        // 15 universal in EUR = 15 / 1.5 = 10
        assertEquals(Integer.valueOf( i: 10), EUR.valueInThisCurrency( amount: 100, SEK));

        // 100 EUR = 150 universal
        // 150 universal in SEK = 150 / 0.15 = 1000
        assertEquals(Integer.valueOf( i: 1000), SEK.valueInThisCurrency( amount: 100, EUR));

        // Test same currency
        assertEquals(Integer.valueOf( i: 100), SEK.valueInThisCurrency( amount: 100, SEK));

        // Test 0
        assertEquals(Integer.valueOf( i: 0), DKK.valueInThisCurrency( amount: 0, SEK));
    }

}
```

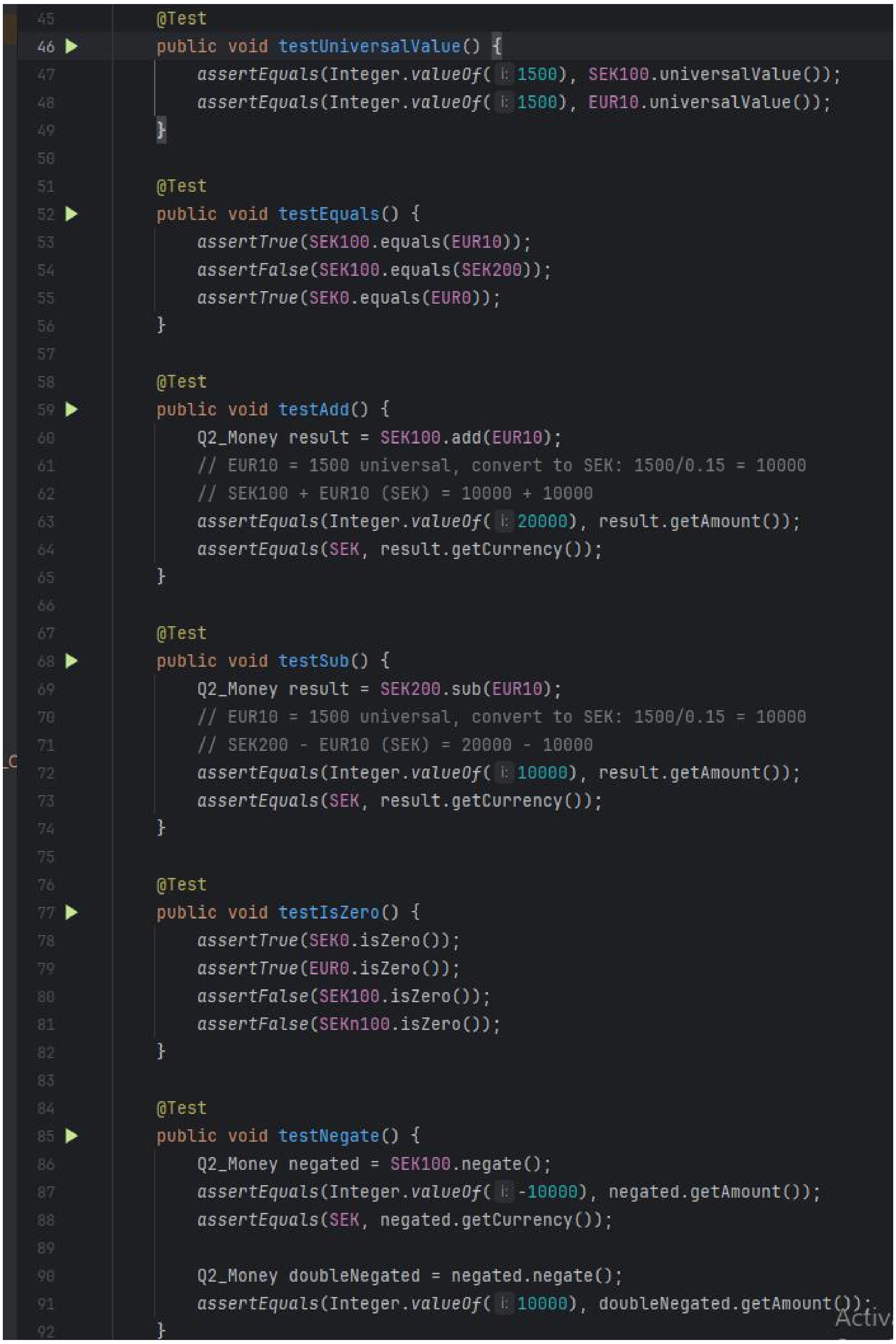
```
Q2_CurrencyTest              1 ms
    testGlobalValue          1 ms
    testGetName              0 ms
    testGetRate              0 ms
    testValueInThisCurren    0 ms
    testSetRate              0 ms
```

```
5 tests passed  5 tests total, 1 ms
C:\Users\Admin\.jdks\ms-17.0.17\bin\java.exe ...

Process finished with exit code 0
```

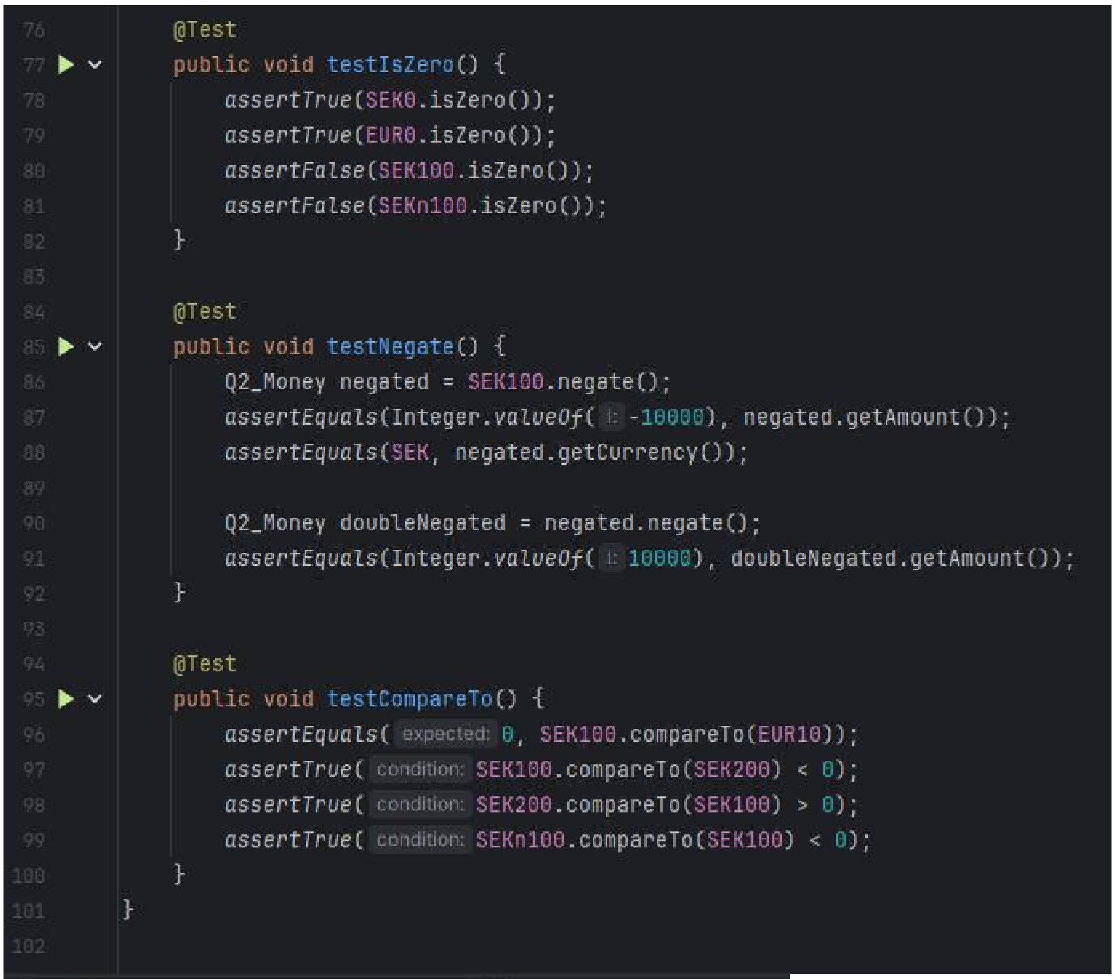
**-Money test:**



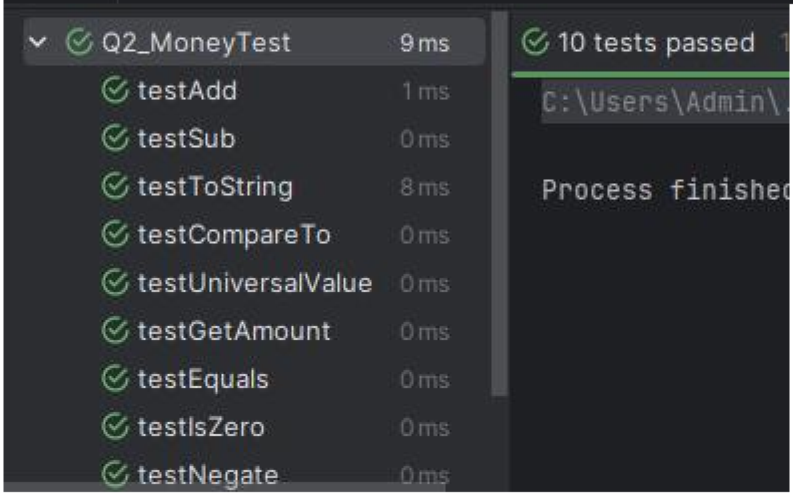
```java
public class Q2_MoneyTest {
    Q2_Currency SEK, DKK, EUR;  9 usages
    Q2_Money SEK100, EUR10, SEK200, EUR20, SEK0, EUR0, SEKn100;  14 usages

    @Before
    public void setUp() throws Exception {
        SEK = new Q2_Currency( name: "SEK",  rate: 0.15);
        DKK = new Q2_Currency( name: "DKK",  rate: 0.20);
        EUR = new Q2_Currency( name: "EUR",  rate: 1.5);
        SEK100 = new Q2_Money( amount: 10000, SEK);
        EUR10 = new Q2_Money( amount: 1000, EUR);
        SEK200 = new Q2_Money( amount: 20000, SEK);
        EUR20 = new Q2_Money( amount: 2000, EUR);
        SEK0 = new Q2_Money( amount: 0, SEK);
        EUR0 = new Q2_Money( amount: 0, EUR);
        SEKn100 = new Q2_Money( amount: -10000, SEK);
    }

    @Test
    public void testGetAmount() {
        assertEquals(Integer.valueOf( i: 10000), SEK100.getAmount());
        assertEquals(Integer.valueOf( i: 1000), EUR10.getAmount());
        assertEquals(Integer.valueOf( i: 0), SEK0.getAmount());
        assertEquals(Integer.valueOf( i: -10000), SEKn100.getAmount());
    }

    @Test
    public void testGetCurrency() {
        assertEquals(SEK, SEK100.getCurrency());
        assertEquals(EUR, EUR10.getCurrency());
    }

    @Test
    public void testToString() {
        assertEquals( expected: "100.0 SEK", SEK100.toString());
        assertEquals( expected: "10.0 EUR", EUR10.toString());
        assertEquals( expected: "0.0 SEK", SEK0.toString());
    }

    @Test
    public void testUniversalValue() {
        assertEquals(Integer.valueOf( i: 1500), SEK100.universalValue());
        assertEquals(Integer.valueOf( i: 1500), EUR10.universalValue());
    }

    @Test
    public void testEquals() {
        assertTrue(SEK100.equals(EUR10));
```



```java
    @Test
    public void testUniversalValue() {
        assertEquals(Integer.valueOf(1500), SEK100.universalValue());
        assertEquals(Integer.valueOf(1500), EUR10.universalValue());
    }

    @Test
    public void testEquals() {
        assertTrue(SEK100.equals(EUR10));
        assertFalse(SEK100.equals(SEK200));
        assertTrue(SEK0.equals(EUR0));
    }

    @Test
    public void testAdd() {
        Q2_Money result = SEK100.add(EUR10);
        // EUR10 = 1500 universal, convert to SEK: 1500/0.15 = 10000
        // SEK100 + EUR10 (SEK) = 10000 + 10000
        assertEquals(Integer.valueOf(20000), result.getAmount());
        assertEquals(SEK, result.getCurrency());
    }

    @Test
    public void testSub() {
        Q2_Money result = SEK200.sub(EUR10);
        // EUR10 = 1500 universal, convert to SEK: 1500/0.15 = 10000
        // SEK200 - EUR10 (SEK) = 20000 - 10000
        assertEquals(Integer.valueOf(10000), result.getAmount());
        assertEquals(SEK, result.getCurrency());
    }

    @Test
    public void testIsZero() {
        assertTrue(SEK0.isZero());
        assertTrue(EUR0.isZero());
        assertFalse(SEK100.isZero());
        assertFalse(SEKn100.isZero());
    }

    @Test
    public void testNegate() {
        Q2_Money negated = SEK100.negate();
        assertEquals(Integer.valueOf(-10000), negated.getAmount());
        assertEquals(SEK, negated.getCurrency());

        Q2_Money doubleNegated = negated.negate();
        assertEquals(Integer.valueOf(10000), doubleNegated.getAmount());
    }
```



```java
    @Test
    public void testIsZero() {
        assertTrue(SEK0.isZero());
        assertTrue(EUR0.isZero());
        assertFalse(SEK100.isZero());
        assertFalse(SEKn100.isZero());
    }

    @Test
    public void testNegate() {
        Q2_Money negated = SEK100.negate();
        assertEquals(Integer.valueOf( i: -10000), negated.getAmount());
        assertEquals(SEK, negated.getCurrency());

        Q2_Money doubleNegated = negated.negate();
        assertEquals(Integer.valueOf( i: 10000), doubleNegated.getAmount());
    }

    @Test
    public void testCompareTo() {
        assertEquals( expected: 0, SEK100.compareTo(EUR10));
        assertTrue( condition: SEK100.compareTo(SEK200) < 0);
        assertTrue( condition: SEK200.compareTo(SEK100) > 0);
        assertTrue( condition: SEKn100.compareTo(SEK100) < 0);
    }
}
```

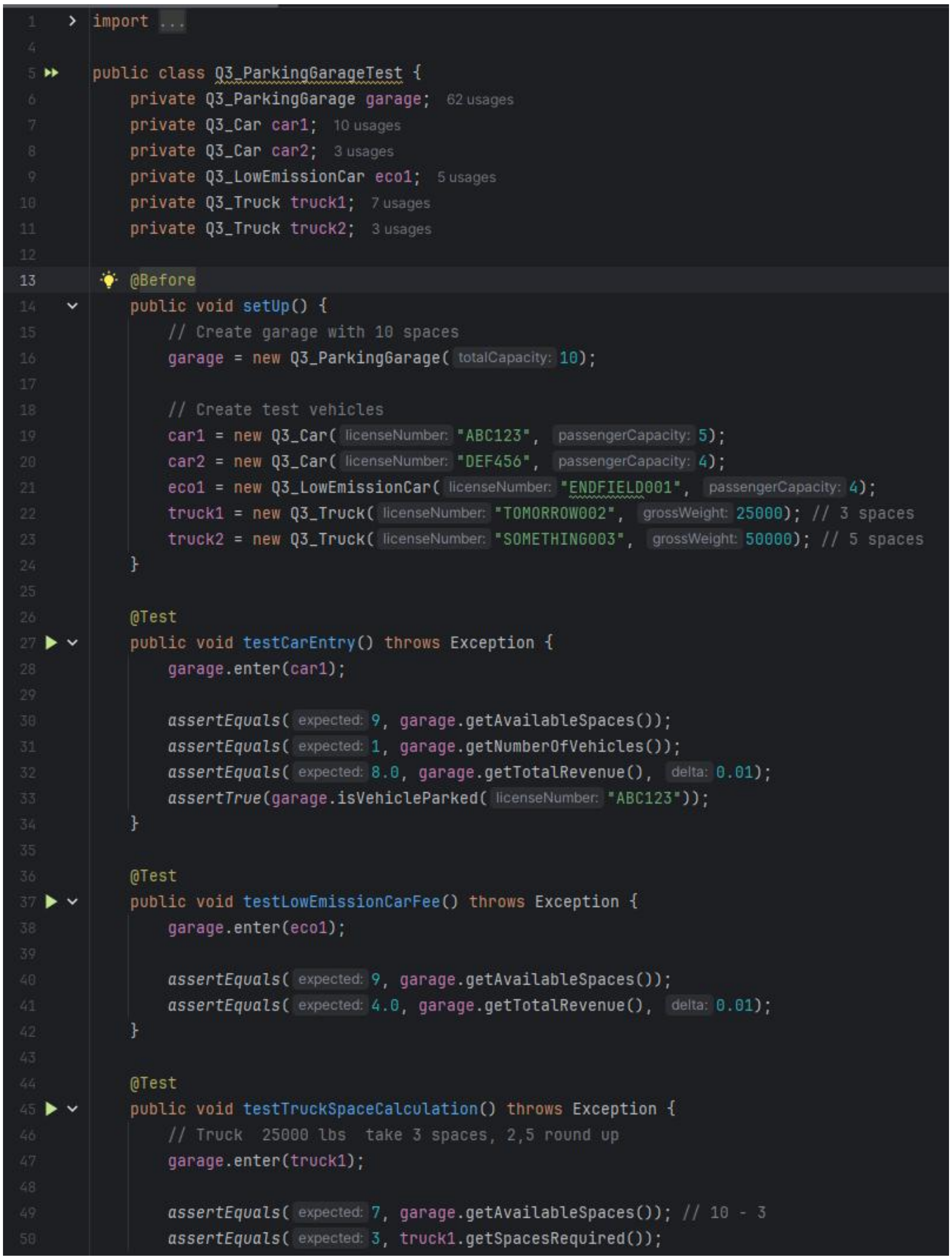


```
Q2_MoneyTest          9 ms     10 tests passed  1
    testAdd           1 ms     C:\Users\Admin\.
    testSub           0 ms
    testToString      8 ms     Process finishe
    testCompareTo     0 ms
    testUniversalValue 0 ms
    testGetAmount     0 ms
    testEquals        0 ms
    testIsZero        0 ms
    testNegate        0 ms
```

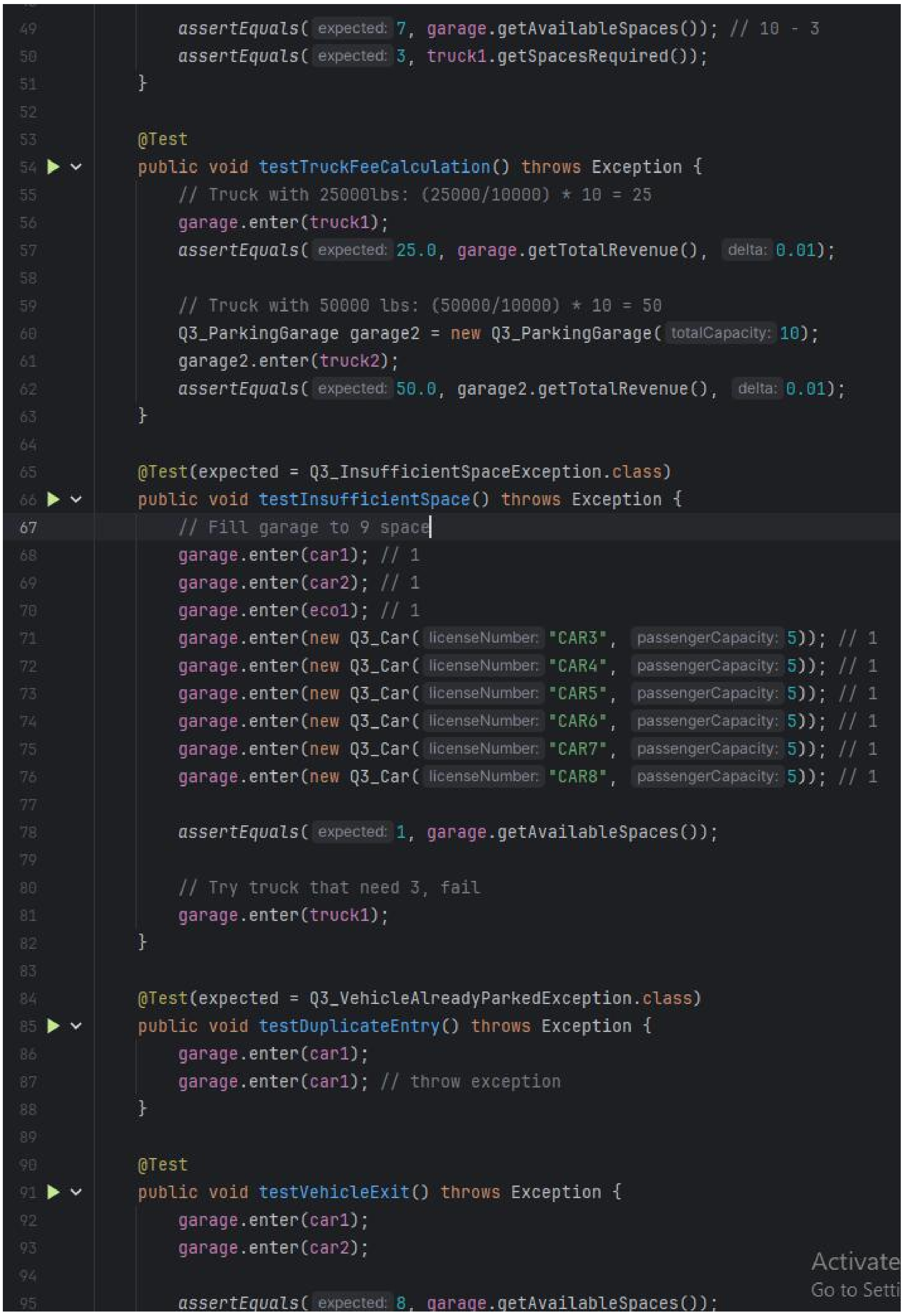
Bonus Points

**3. A parking garage has a number of parking spaces for vehicles. The parking garage must keep track of the vehicles that are currently parked there so that it can report the number of parking spaces available and the current value. A parking garage must provide publicly available functions that are called when a vehicle enters and exits the garage. It can throw exceptions if there is insufficient available space in the garage, if a vehicle tries to enter that is already in the garage, or if a car that is not in the garage tries to exit.**

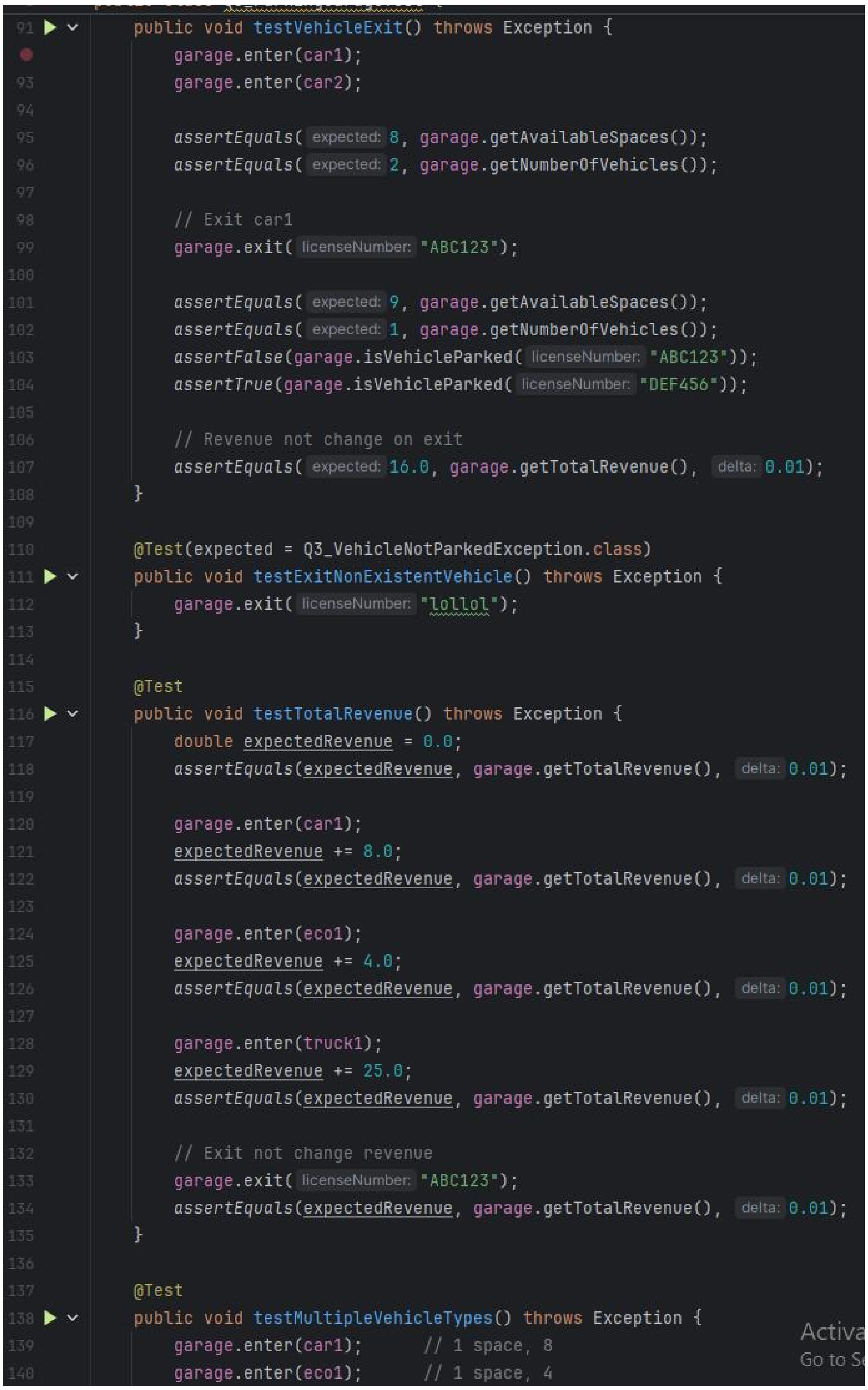
Additionally, the garage must provide publicly available functions that report the total capacity of the
garage, the number of available spaces in the garage, and the total money collected. The parking fee is assessed when a vehicle enters the parking garage.
Currently, there are three distinct types of vehicles: cars, low-emission cars, and cargo trucks. Every vehicle has a license number consisting of letters and/or numbers and the number of parking spaces it occupies. Cars have a passenger capacity (i.e., the number of passengers). Trucks have a gross vehicle weight it can transport. Cars take up one space, while trucks take up a number of spaces that is their gross weight divided by 10,000. Cars are charged a rate of $8, low-emission cars are charged half that rate, and trucks are charged $10 per 10,000 pounds of gross weight.Develop a software based on TDD to solve the above problem.60 Points



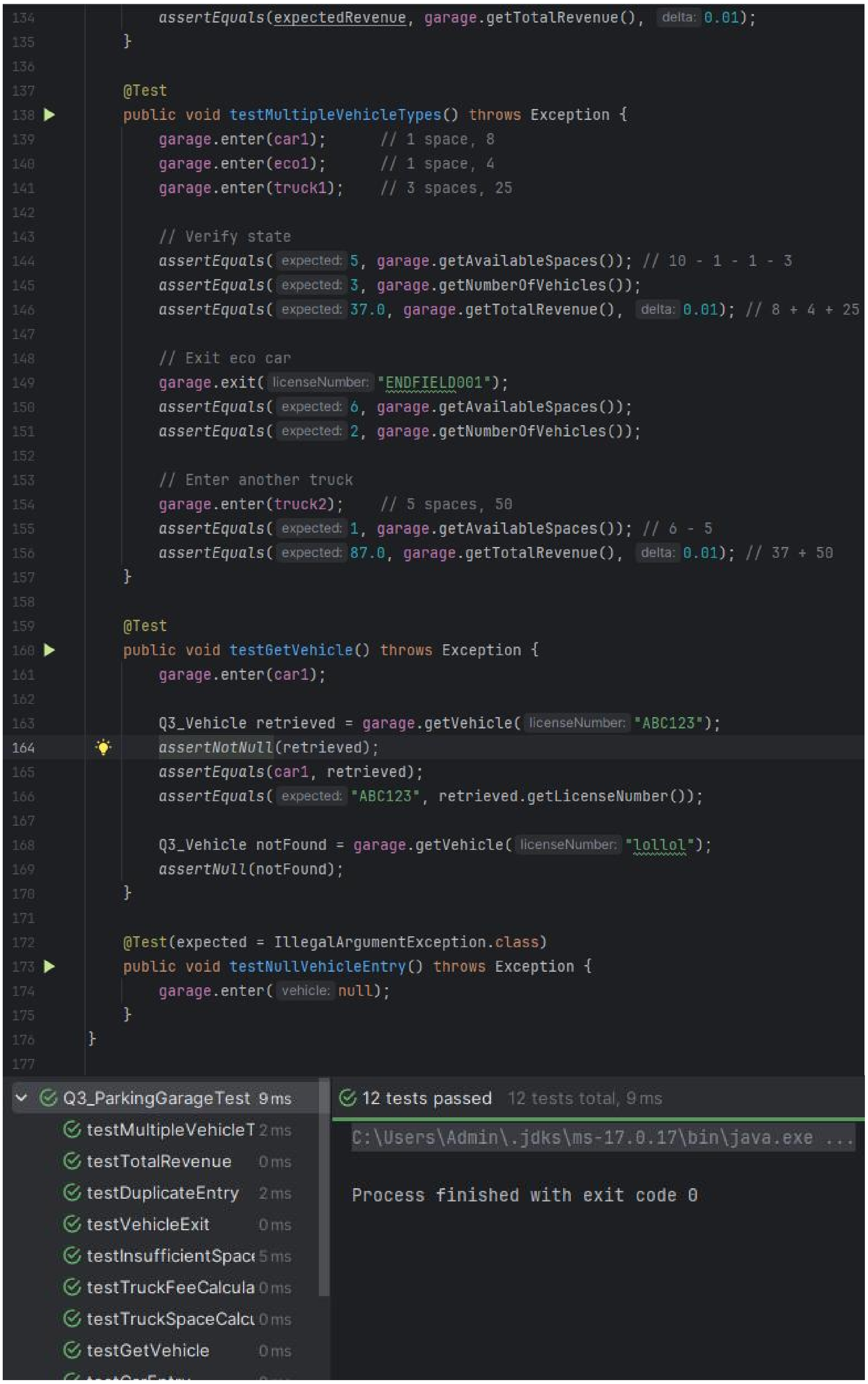
```java
import ...

public class Q3_ParkingGarageTest {
    private Q3_ParkingGarage garage;  62 usages
    private Q3_Car car1;  10 usages
    private Q3_Car car2;  3 usages
    private Q3_LowEmissionCar eco1;  5 usages
    private Q3_Truck truck1;  7 usages
    private Q3_Truck truck2;  3 usages

    @Before
    public void setUp() {
        // Create garage with 10 spaces
        garage = new Q3_ParkingGarage( totalCapacity: 10);

        // Create test vehicles
        car1 = new Q3_Car( licenseNumber: "ABC123",  passengerCapacity: 5);
        car2 = new Q3_Car( licenseNumber: "DEF456",  passengerCapacity: 4);
        eco1 = new Q3_LowEmissionCar( licenseNumber: "ENDFIELD001",  passengerCapacity: 4);
        truck1 = new Q3_Truck( licenseNumber: "TOMORROW002",  grossWeight: 25000); // 3 spaces
        truck2 = new Q3_Truck( licenseNumber: "SOMETHING003",  grossWeight: 50000); // 5 spaces
    }

    @Test
    public void testCarEntry() throws Exception {
        garage.enter(car1);

        assertEquals( expected: 9, garage.getAvailableSpaces());
        assertEquals( expected: 1, garage.getNumberOfVehicles());
        assertEquals( expected: 8.0, garage.getTotalRevenue(),  delta: 0.01);
        assertTrue(garage.isVehicleParked( licenseNumber: "ABC123"));
    }

    @Test
    public void testLowEmissionCarFee() throws Exception {
        garage.enter(eco1);

        assertEquals( expected: 9, garage.getAvailableSpaces());
        assertEquals( expected: 4.0, garage.getTotalRevenue(),  delta: 0.01);
    }

    @Test
    public void testTruckSpaceCalculation() throws Exception {
        // Truck  25000 lbs  take 3 spaces, 2,5 round up
        garage.enter(truck1);

        assertEquals( expected: 7, garage.getAvailableSpaces()); // 10 - 3
        assertEquals( expected: 3, truck1.getSpacesRequired());
```



```java
        assertEquals( expected: 7, garage.getAvailableSpaces()); // 10 - 3
        assertEquals( expected: 3, truck1.getSpacesRequired());
    }

    @Test
    public void testTruckFeeCalculation() throws Exception {
        // Truck with 25000lbs: (25000/10000) * 10 = 25
        garage.enter(truck1);
        assertEquals( expected: 25.0, garage.getTotalRevenue(),  delta: 0.01);

        // Truck with 50000 lbs: (50000/10000) * 10 = 50
        Q3_ParkingGarage garage2 = new Q3_ParkingGarage( totalCapacity: 10);
        garage2.enter(truck2);
        assertEquals( expected: 50.0, garage2.getTotalRevenue(),  delta: 0.01);
    }

    @Test(expected = Q3_InsufficientSpaceException.class)
    public void testInsufficientSpace() throws Exception {
        // Fill garage to 9 space
        garage.enter(car1); // 1
        garage.enter(car2); // 1
        garage.enter(eco1); // 1
        garage.enter(new Q3_Car( licenseNumber: "CAR3",  passengerCapacity: 5)); // 1
        garage.enter(new Q3_Car( licenseNumber: "CAR4",  passengerCapacity: 5)); // 1
        garage.enter(new Q3_Car( licenseNumber: "CAR5",  passengerCapacity: 5)); // 1
        garage.enter(new Q3_Car( licenseNumber: "CAR6",  passengerCapacity: 5)); // 1
        garage.enter(new Q3_Car( licenseNumber: "CAR7",  passengerCapacity: 5)); // 1
        garage.enter(new Q3_Car( licenseNumber: "CAR8",  passengerCapacity: 5)); // 1

        assertEquals( expected: 1, garage.getAvailableSpaces());

        // Try truck that need 3, fail
        garage.enter(truck1);
    }

    @Test(expected = Q3_VehicleAlreadyParkedException.class)
    public void testDuplicateEntry() throws Exception {
        garage.enter(car1);
        garage.enter(car1); // throw exception
    }

    @Test
    public void testVehicleExit() throws Exception {
        garage.enter(car1);
        garage.enter(car2);

        assertEquals( expected: 8, garage.getAvailableSpaces());
```



```java
    public void testVehicleExit() throws Exception {
        garage.enter(car1);
        garage.enter(car2);

        assertEquals( expected: 8, garage.getAvailableSpaces());
        assertEquals( expected: 2, garage.getNumberOfVehicles());

        // Exit car1
        garage.exit( licenseNumber: "ABC123");

        assertEquals( expected: 9, garage.getAvailableSpaces());
        assertEquals( expected: 1, garage.getNumberOfVehicles());
        assertFalse(garage.isVehicleParked( licenseNumber: "ABC123"));
        assertTrue(garage.isVehicleParked( licenseNumber: "DEF456"));

        // Revenue not change on exit
        assertEquals( expected: 16.0, garage.getTotalRevenue(),  delta: 0.01);
    }

    @Test(expected = Q3_VehicleNotParkedException.class)
    public void testExitNonExistentVehicle() throws Exception {
        garage.exit( licenseNumber: "lollol");
    }

    @Test
    public void testTotalRevenue() throws Exception {
        double expectedRevenue = 0.0;
        assertEquals(expectedRevenue, garage.getTotalRevenue(),  delta: 0.01);

        garage.enter(car1);
        expectedRevenue += 8.0;
        assertEquals(expectedRevenue, garage.getTotalRevenue(),  delta: 0.01);

        garage.enter(eco1);
        expectedRevenue += 4.0;
        assertEquals(expectedRevenue, garage.getTotalRevenue(),  delta: 0.01);

        garage.enter(truck1);
        expectedRevenue += 25.0;
        assertEquals(expectedRevenue, garage.getTotalRevenue(),  delta: 0.01);

        // Exit not change revenue
        garage.exit( licenseNumber: "ABC123");
        assertEquals(expectedRevenue, garage.getTotalRevenue(),  delta: 0.01);
    }

    @Test
    public void testMultipleVehicleTypes() throws Exception {
        garage.enter(car1);      // 1 space, 8
        garage.enter(eco1);      // 1 space, 4
```



```java
        assertEquals(expectedRevenue, garage.getTotalRevenue(),  delta: 0.01);
    }

    @Test
    public void testMultipleVehicleTypes() throws Exception {
        garage.enter(car1);      // 1 space, 8
        garage.enter(eco1);      // 1 space, 4
        garage.enter(truck1);    // 3 spaces, 25

        // Verify state
        assertEquals( expected: 5, garage.getAvailableSpaces()); // 10 - 1 - 1 - 3
        assertEquals( expected: 3, garage.getNumberOfVehicles());
        assertEquals( expected: 37.0, garage.getTotalRevenue(),  delta: 0.01); // 8 + 4 + 25

        // Exit eco car
        garage.exit( licenseNumber: "ENDFIELD001");
        assertEquals( expected: 6, garage.getAvailableSpaces());
        assertEquals( expected: 2, garage.getNumberOfVehicles());

        // Enter another truck
        garage.enter(truck2);    // 5 spaces, 50
        assertEquals( expected: 1, garage.getAvailableSpaces()); // 6 - 5
        assertEquals( expected: 87.0, garage.getTotalRevenue(),  delta: 0.01); // 37 + 50
    }

    @Test
    public void testGetVehicle() throws Exception {
        garage.enter(car1);

        Q3_Vehicle retrieved = garage.getVehicle( licenseNumber: "ABC123");
        assertNotNull(retrieved);
        assertEquals(car1, retrieved);
        assertEquals( expected: "ABC123", retrieved.getLicenseNumber());

        Q3_Vehicle notFound = garage.getVehicle( licenseNumber: "lollol");
        assertNull(notFound);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testNullVehicleEntry() throws Exception {
        garage.enter( vehicle: null);
    }
}
```

```
Q3_ParkingGarageTest 9 ms        12 tests passed   12 tests total, 9 ms
  testMultipleVehicleT 2 ms      C:\Users\Admin\.jdks\ms-17.0.17\bin\java.exe ...
  testTotalRevenue     0 ms
  testDuplicateEntry   2 ms      Process finished with exit code 0
  testVehicleExit      0 ms
  testInsufficientSpac 5 ms
  testTruckFeeCalcula  0 ms
  testTruckSpaceCalcu  0 ms
  testGetVehicle       0 ms
```