**Testing for a**

**Bank system**

**Hadeel, Rehan, Tawadod**

**1\_ Contant:**

1. **Content**
2. **Description**
3. **Data**
4. **Test methodology**
   1. Environment (HW, SW)
   2. Tools for test
5. **Black-Box:**
   1. Manual Testing
      1. Equivalence Partitioning
      2. Decision Tables
      3. Boundary Value Analysis
      4. State Transition Testing
      5. Use Case Testing
   2. System Test
   3. Acceptance Test
6. **White Box:**

6.1. Manual

* + 1. Flow chart
    2. Coverage table
    3. Coverage percentage

6.2. Junit coverage

**7** Uncovered **Tests**

**2\_ Descriptions:**

The bank system is a simple system that allows the user who has a bank account to use basic transactions such as withdrawal and deposit and inquire about his balance, taking into account the security conditions such as matching the password.

**3\_ Data**

|  |  |  |
| --- | --- | --- |
| **Author** | **Date** | **Test** |
| **Hadeel AL Qadhi** | **31/12/2023** | **Black box** |
| **Rehan AL Salami** | **21/1/2024** | **white box** |
| **Tawadod AL Zeki** | **2/2/2024** | **Finalizing** |

**4\_ Test methodology**

**Environment:**

**Hardware : laptop core i5**

**Software : windows operating system , java editor**

**Testing tools : Junit**

**The test dose not need a test group or a specific test group .**

**5. Black-Box:**

**5.1. Manual Testing**

**5.1.1Equivalence Partitioning**

Password field accepts minimum 3 characters and

maximum characters

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Test scenario description** | **Test scenario** |
| Not accepted | From 0-2  (54) | 1 |
| Accepted | From 3-8  (6754) | 2 |
| Not accepted | From 9-12  (89763345326) | 3 |

**5.1.2.Decision tables**

Create the decision table for a deposit method that asks for name and Password.

The condition here is that the user will accepted his deposit

Request.

if he enters the correct user name and password, or not if the input is wring

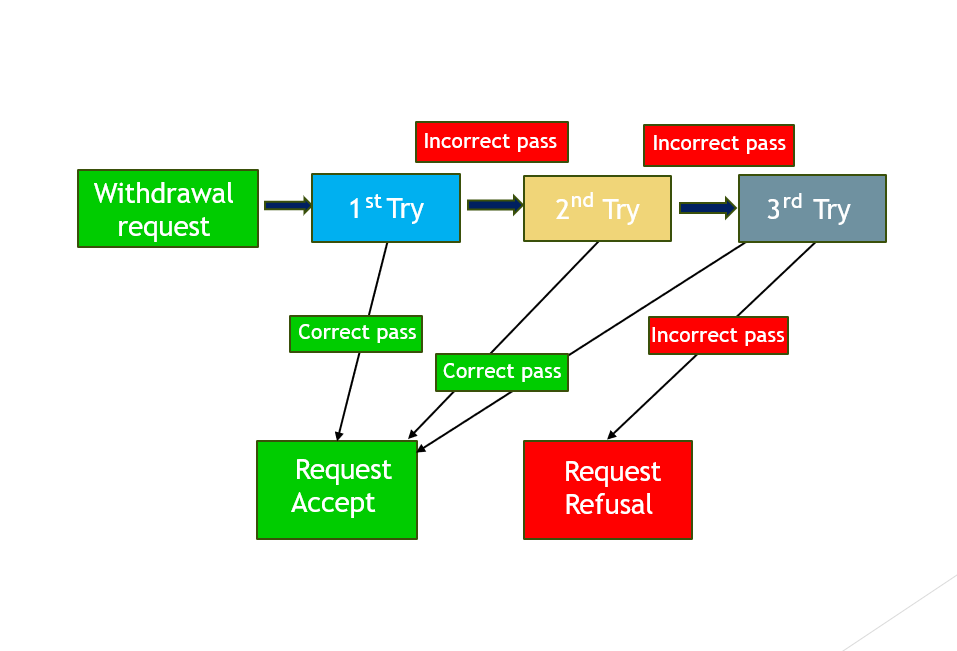
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rule4** | **Rule3** | **Rule2** | **Rule1** | **Condition** |
| T | F | T | F | Name (T, F) |
| T | T | F | F | Password(T,F) |
| Y | N | N | N | Output (Y, N) |

**5.1.3 .Boundary Value Analysis**

withdraw should accept the amount 500 to 5000

|  |  |
| --- | --- |
| **outcome** | **Test scenario description** |
| Not accepted | Boundary=70 |
| Accepted | Boundary=500 |
| Accepted | Boundary=800 |
| Accepted | Boundary=1000 |
| Accepted | Boundary=3000 |
| Accepted | Boundary=5000 |
| Not accepted | Boundary=6500 |
| Not accepted | Boundary=9000 |

**5.1.4. State transition testing**

**5.1.5. Use Case Testing**

ا.

1\_ يقوم العميل بتحديد عملية السحب.

2\_ يقوم العميل بإدخال المبلغ المراد سحبه.

3\_ يقوم النظام بالتحقق من صحة المعلومات المدخلة ومن توفر الرصيد الكافي في حساب العميل

4\_ إذا تم التحقق بنجاح، يقوم النظام بتنفيذ العملية وتحديث رصيد الحساب. ب.

1\_ يقوم العميل بتحديد عملية الايداع.

2\_ يقوم العميل بإدخال المبلغ المراد ايداعه.

3 \_ يقوم النظام بالتحقق من صحة المعلومات المدخلة ومن توفر الرصيد الكافي في حساب العميل

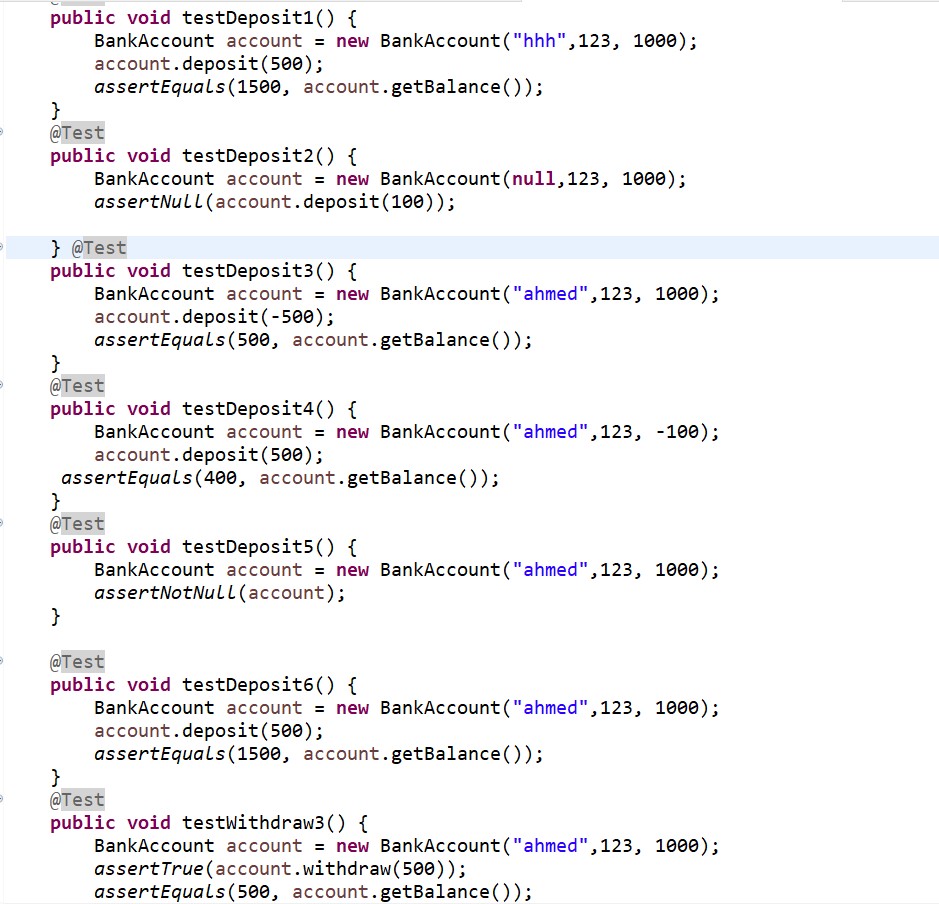
4\_ إذا تم التحقق بنجاح، يقوم النظام بتنفيذ العملية وتحديث رصيد الحساب

5.2. System Test

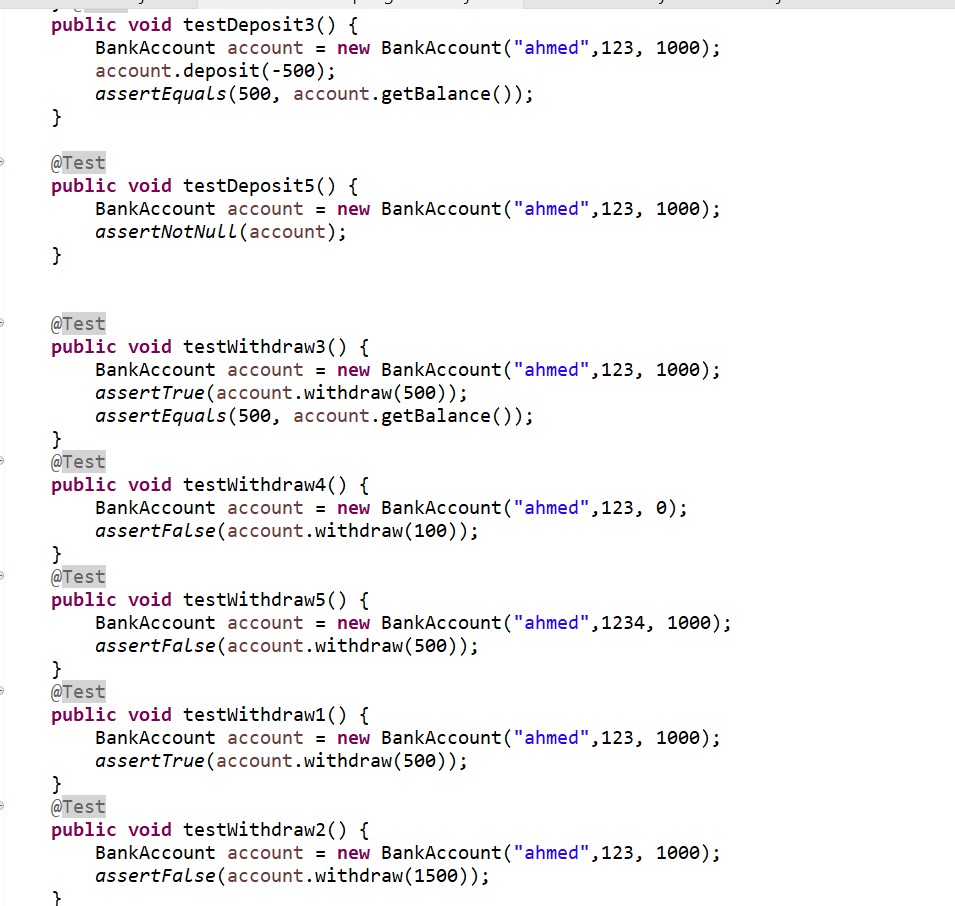
1. **Assert Equals:**

**Case1: checking the balance after adding an amount .**

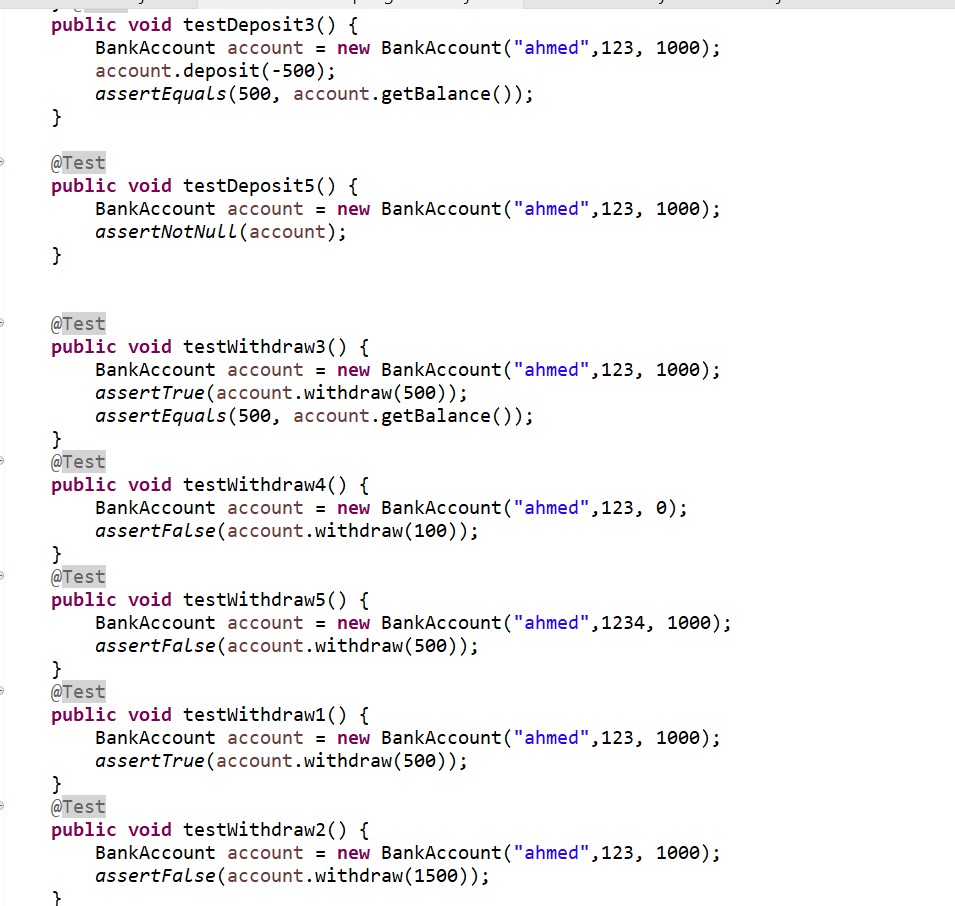
**(Importance :** **Medium)**



**Case2: checking the balance after negative an amount . (Importance :** **Medium)**



**Case3:** **checking the balance after adding an amount . (Importance :** **Low)**

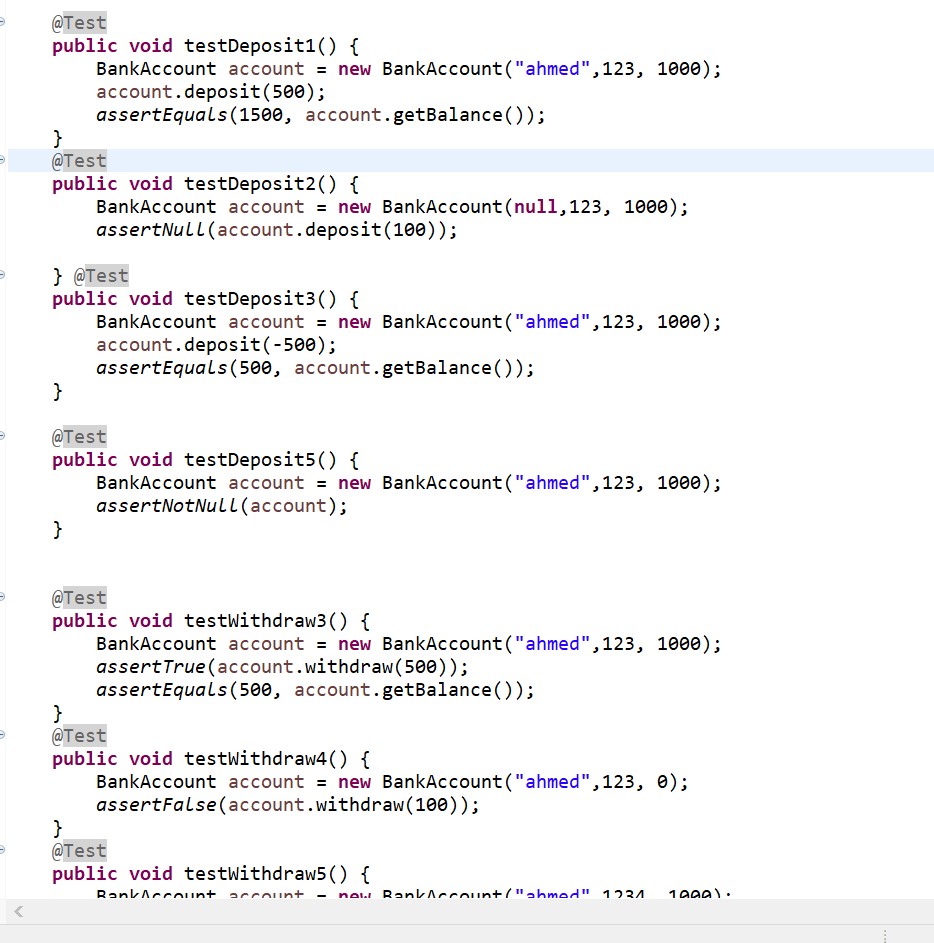


**Case4 :** **checking the balance after adding an amount with error password . (Importance :** **Medium)**



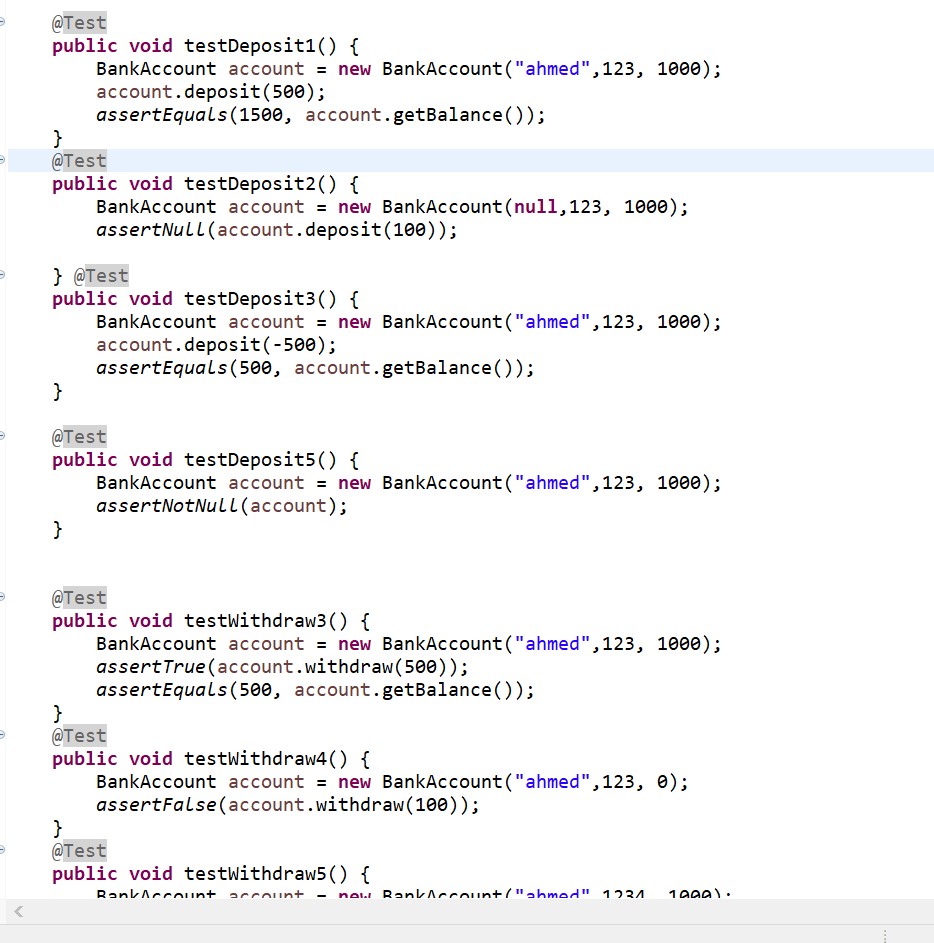
1. **Assert Null:**

**test that if deposit method return null.** **(Importance: Medium)**



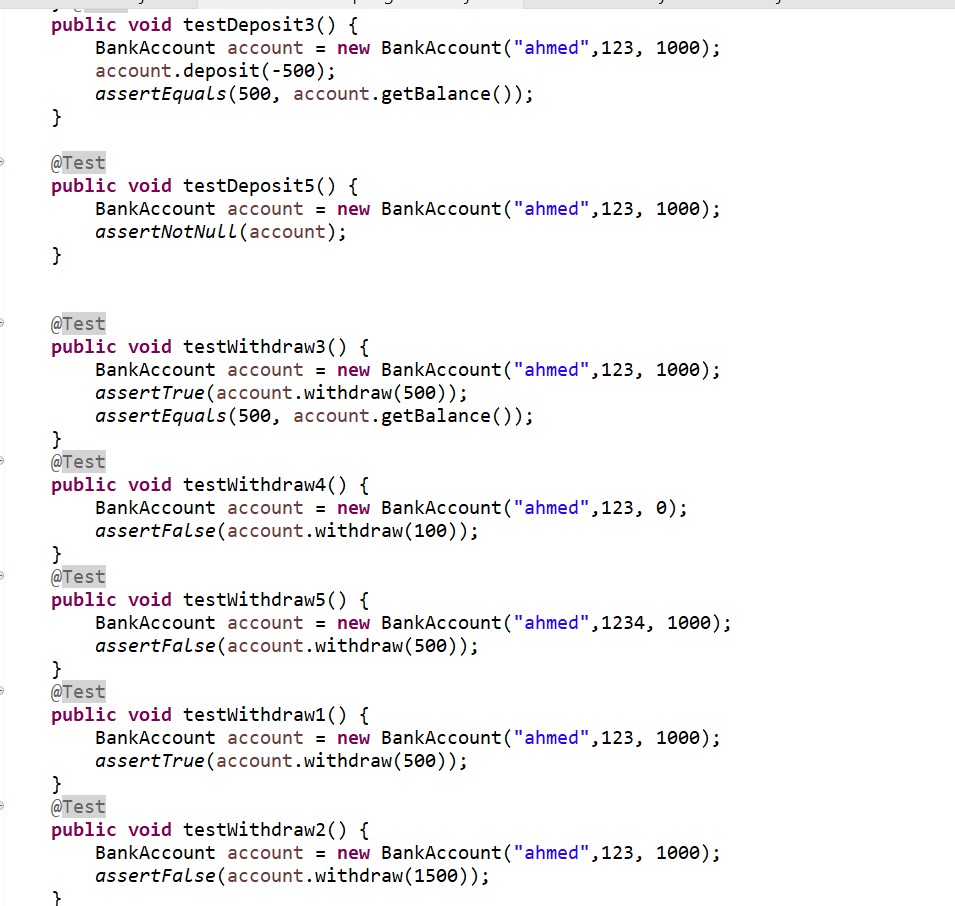
1. **Assert Not Null:**

**test that if there is an account bank . (Importance: Low)**



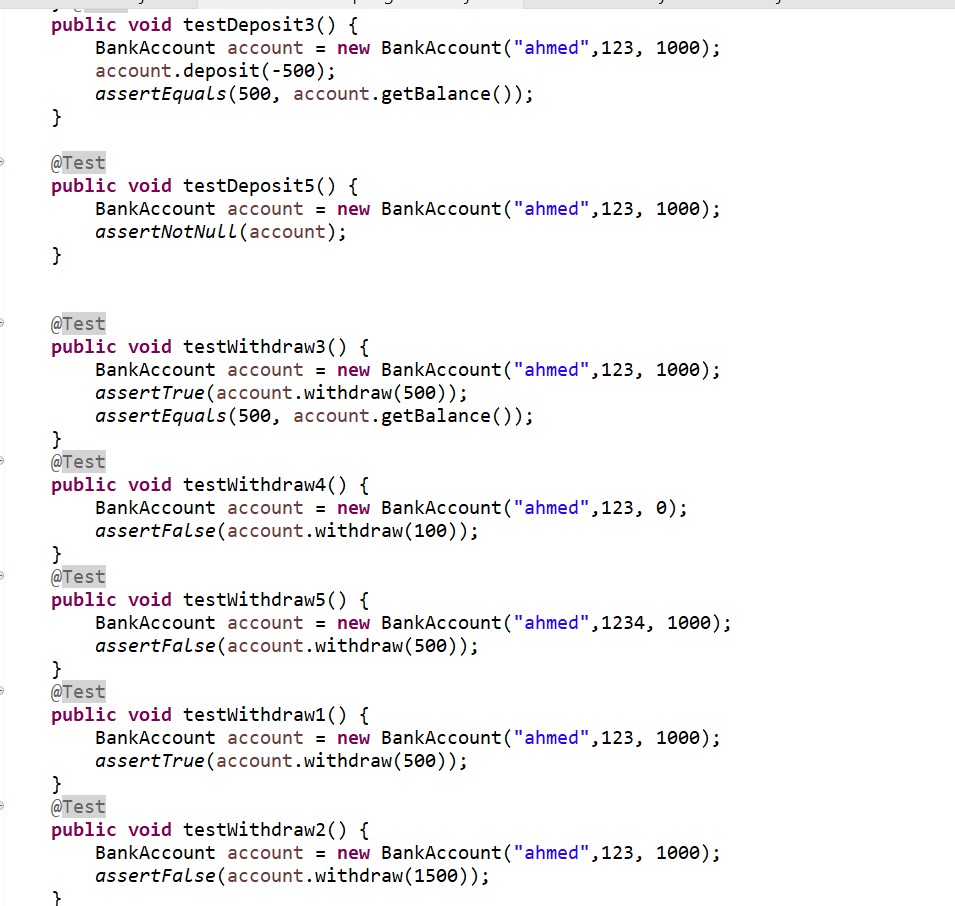
1. **Assert True :**

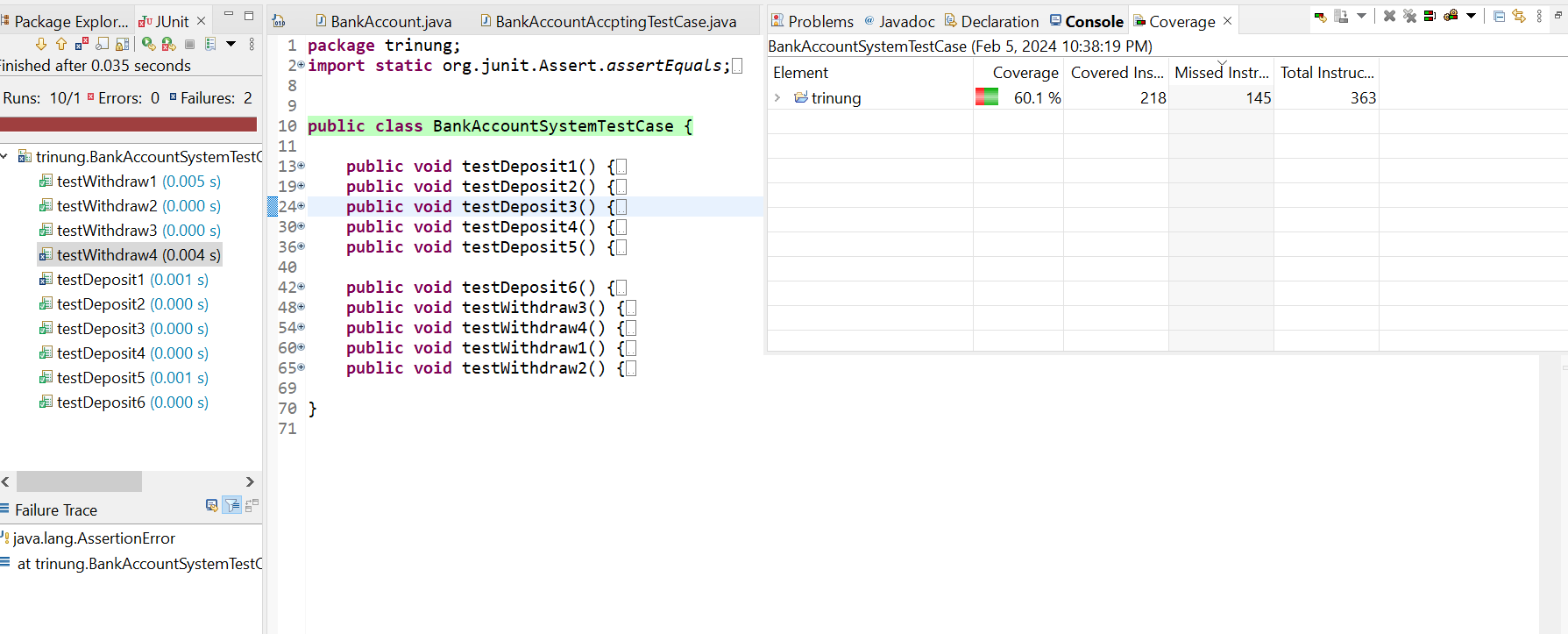
**test that if withdraw method return true** **when the amount match the correct boundary** . **(Importance :** **High)**



1. **Assert False :**

**Case1: test that if withdraw method return False when the amount doesn’t match the correct boundary . (Importance: High)**



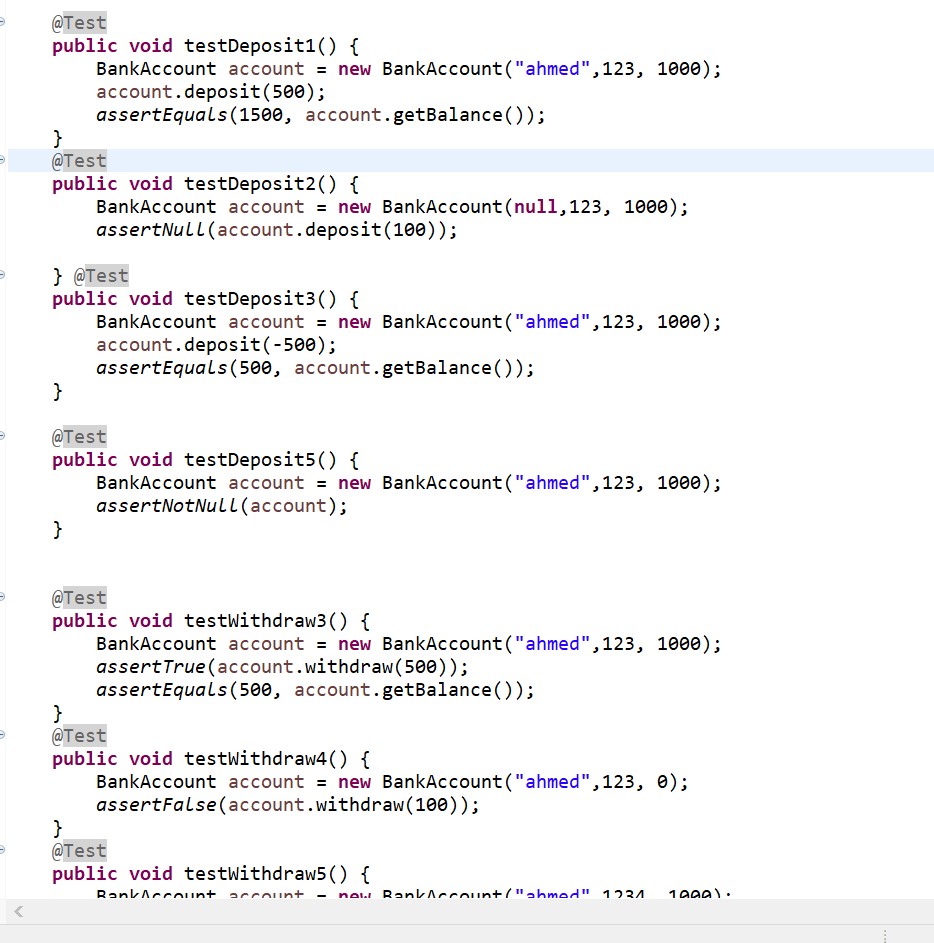


5.3 **Acceptance Test**

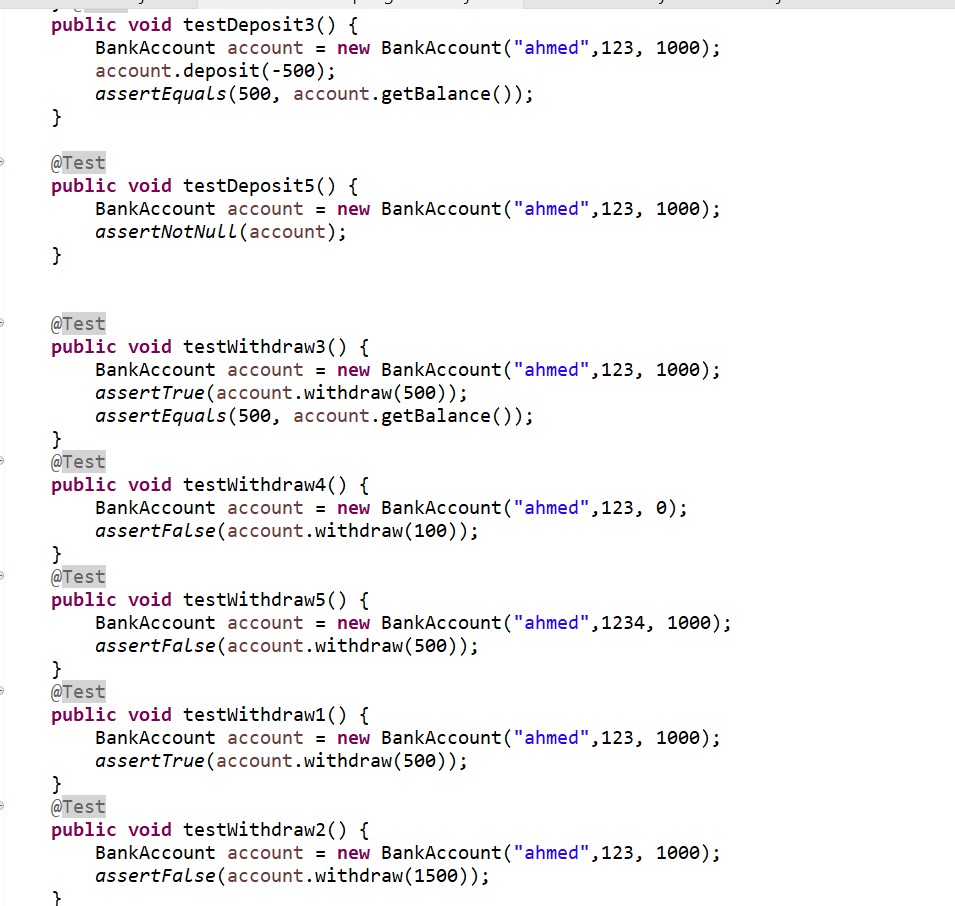
1. **Assert Equals:**

**checking the balance after adding an amount .**

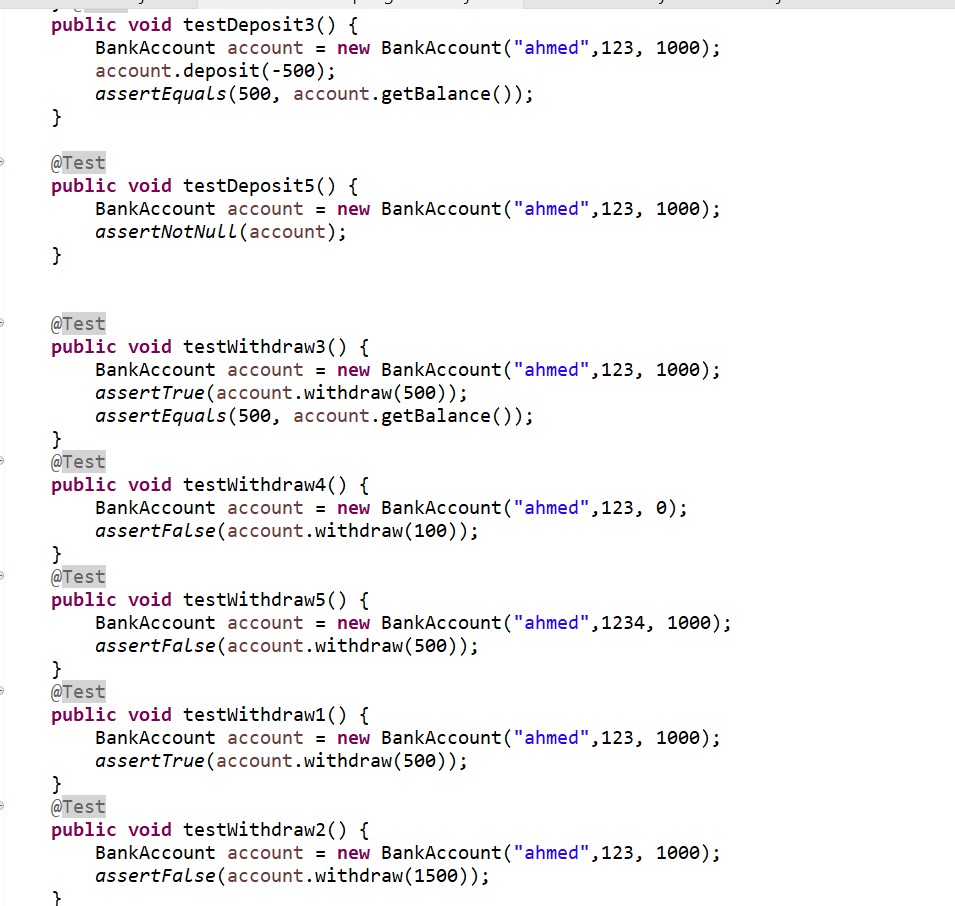
**(Importance :** **Medium)**



Case2: checking the balance after negative an amount . **(Importance :** **Medium)**

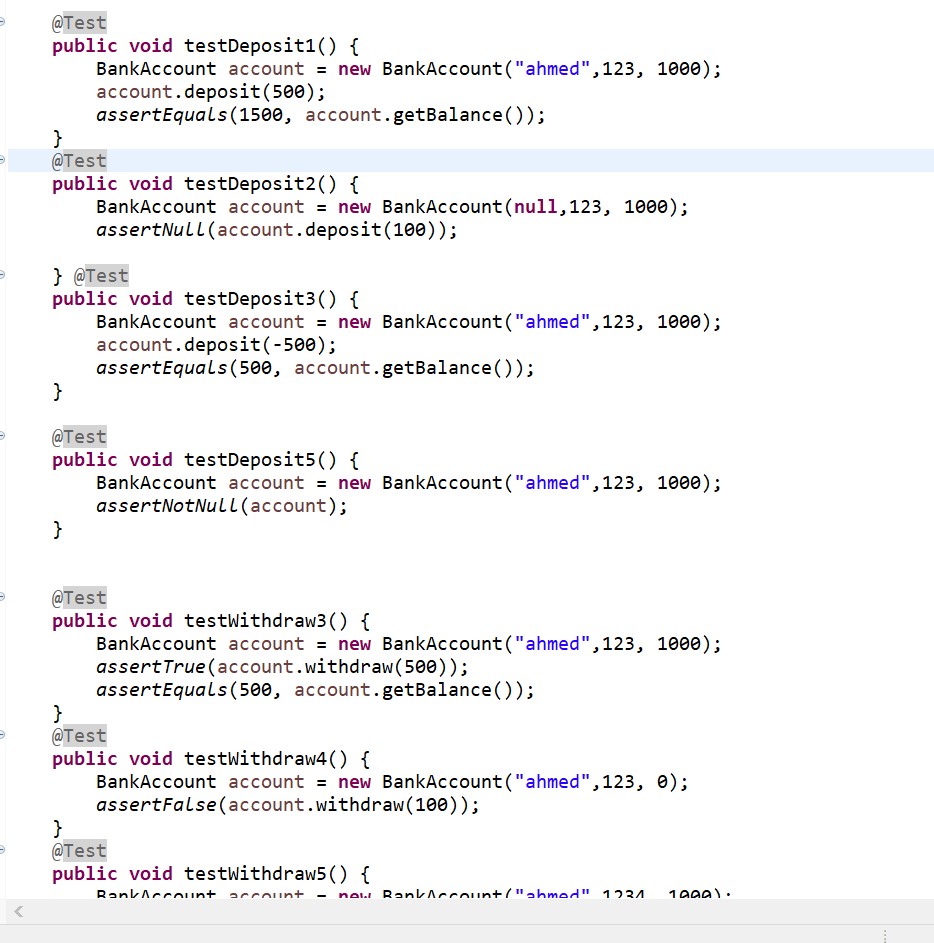


**Case3**: checking the balance after adding an amount . **(Importance :** **Medium)**



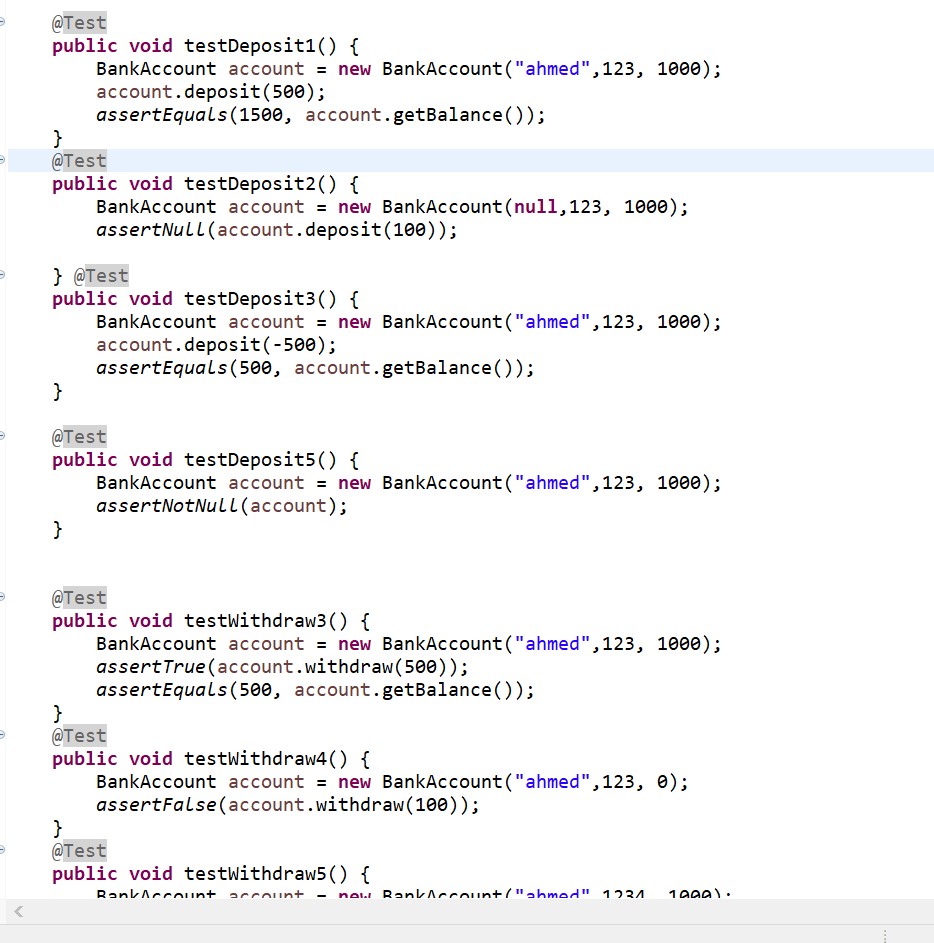
1. **Assert Null:**

**test that if deposit method return null.** **(Importance: Medium)**



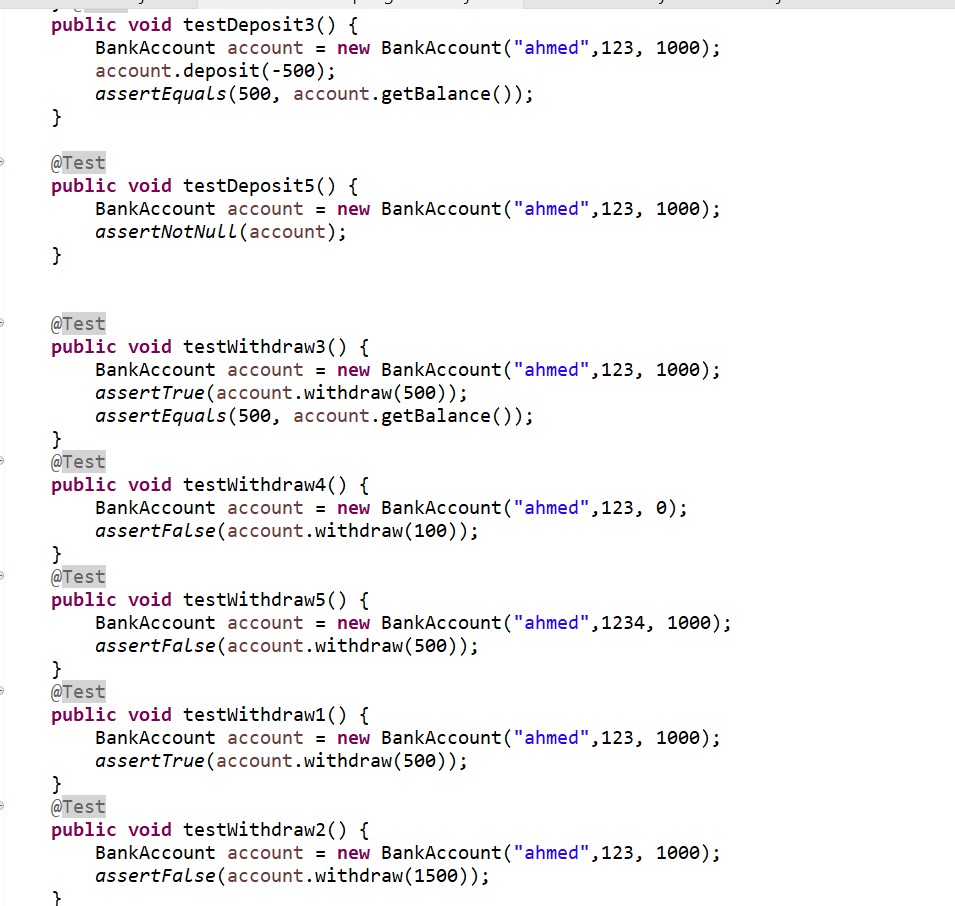
1. **Assert Not Null:**

**test that if there is an account bank . (Importance: Low)**



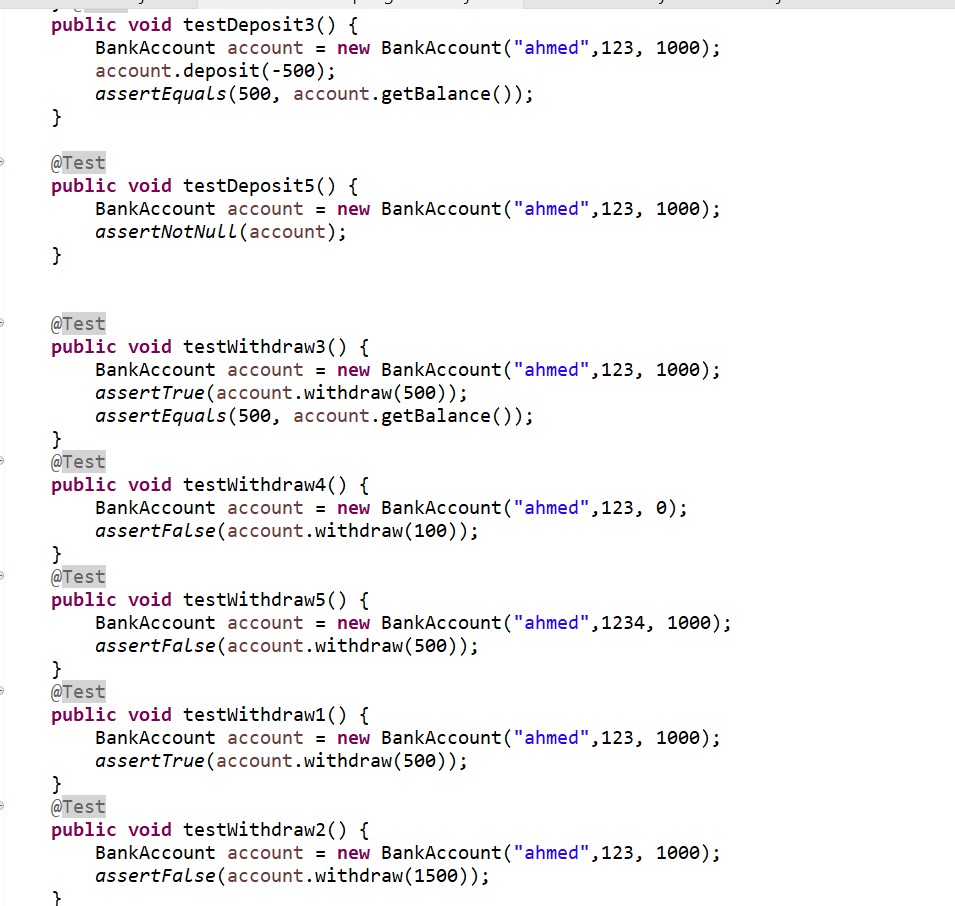
1. **Assert True :**

**test that if withdraw method return true** **when the amount match the correct boundary** . **(Importance :** **High)**

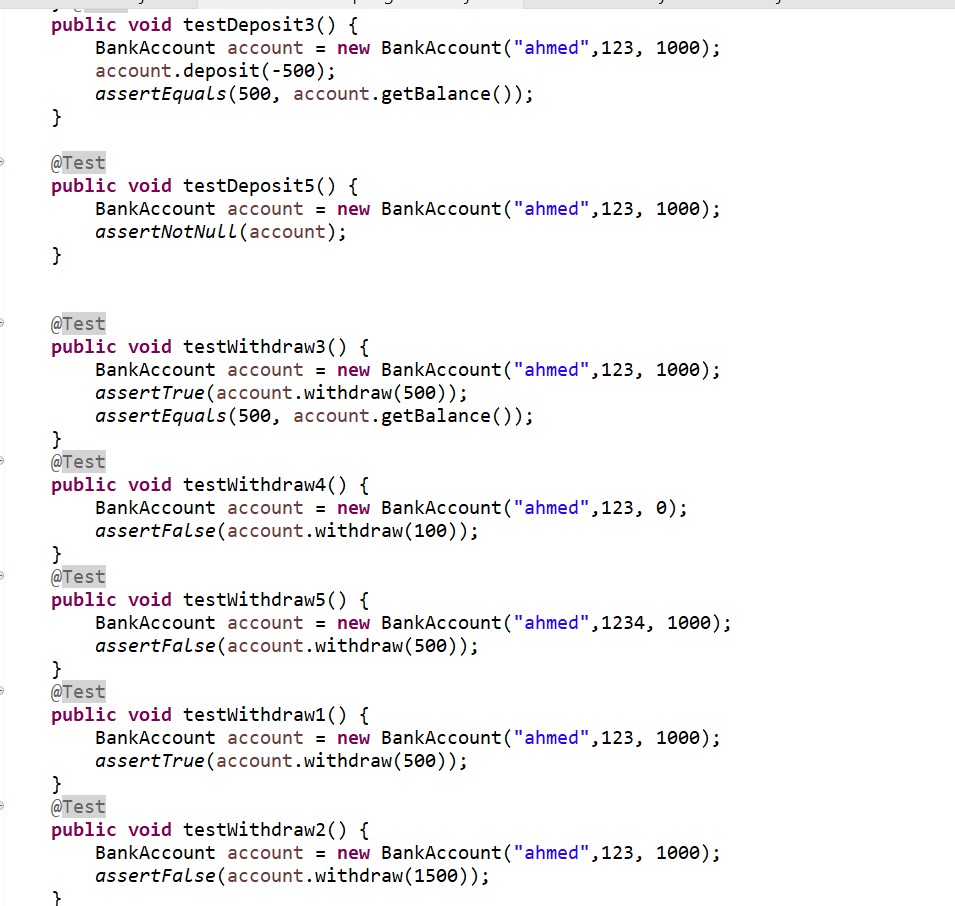


1. **Assert False :**

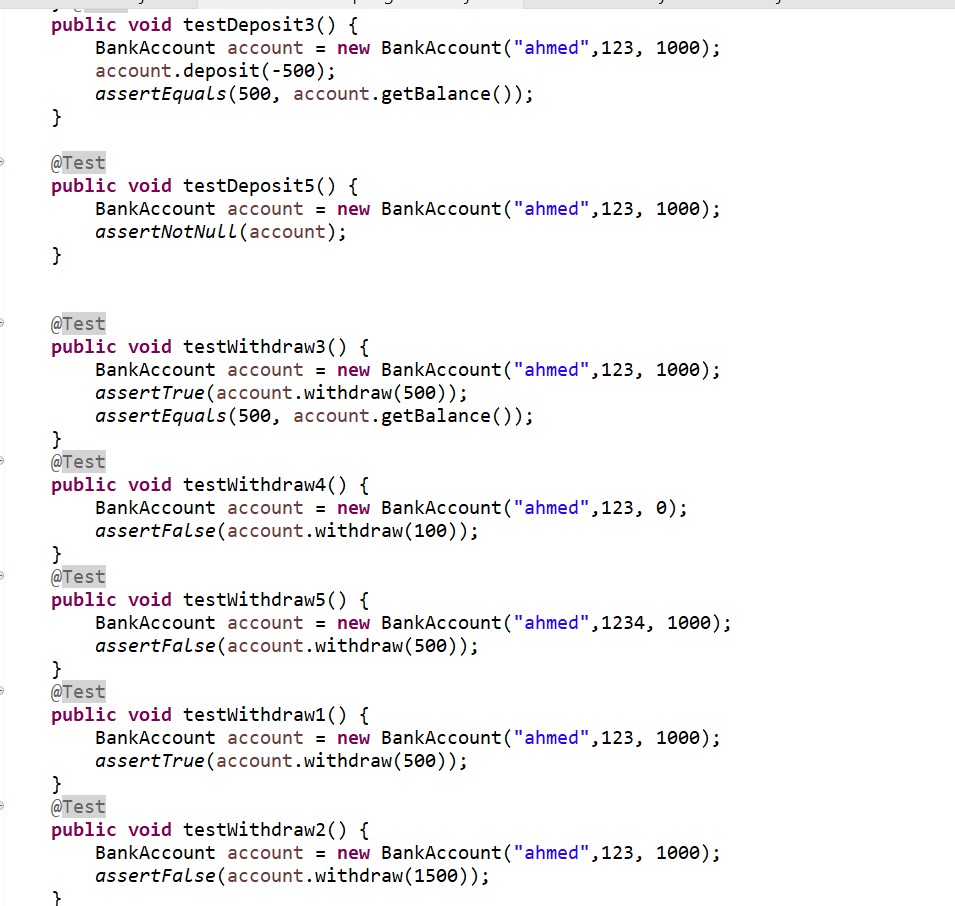
**Case1:** **test that if withdraw method return False when the amount doesn’t match the correct boundary . (Importance: High)**

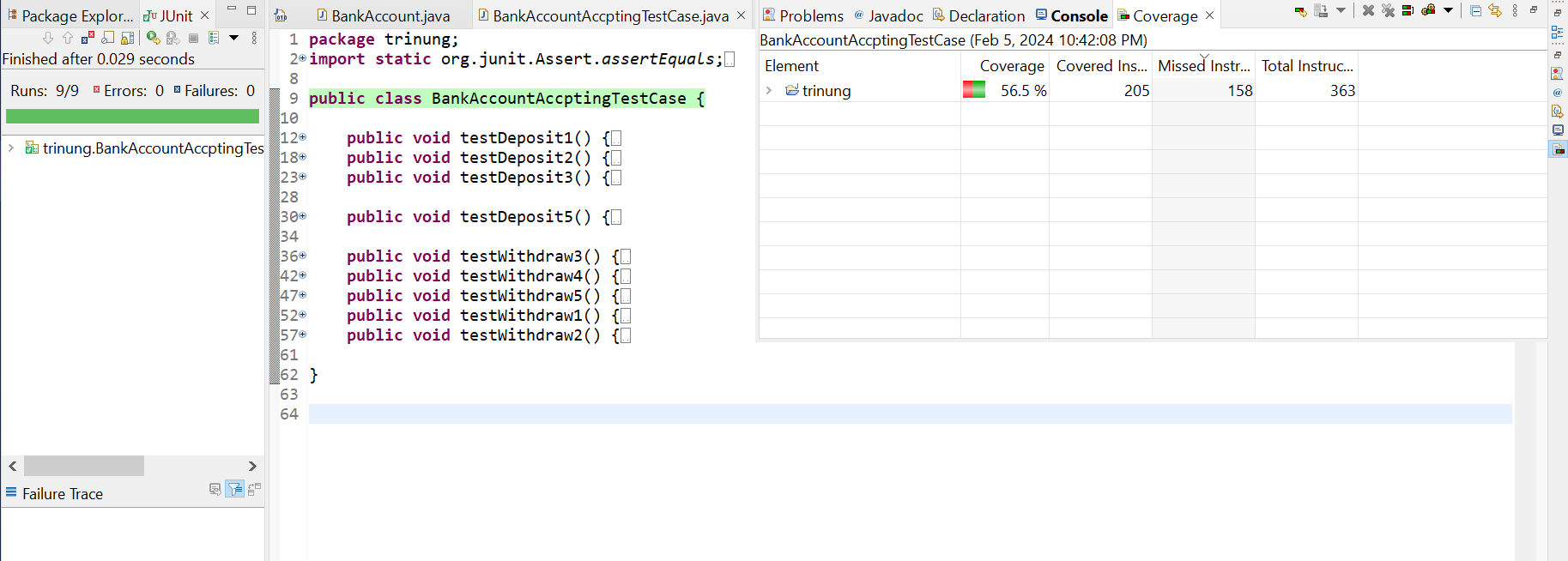


Case2: test that if withdraw method return False when the amount match the correct boundary with a ronge password. **(Importance: High)**



**Case3:** test that if withdraw method return False when the amount match the correct boundary . **(Importance: low)**

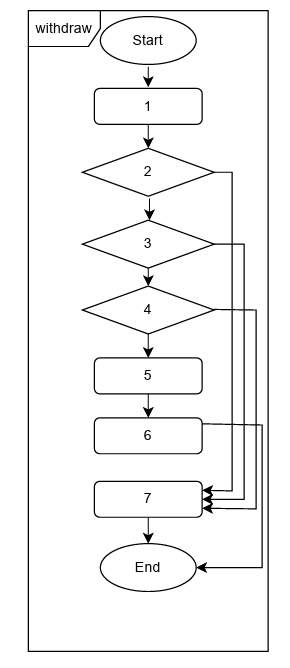




**6. White Box:**

**6.1. Manual**

* + 1. Flow chart
    2. Coverage table
    3. Coverage percentage



**public** **Boolean** withdraw(**int** amount) {1

**if** (owner =="Ahmed"&& password ==123) {2

**if** (500 <= amount && amount <= 5000) {3

**if** (amount <= balance)4

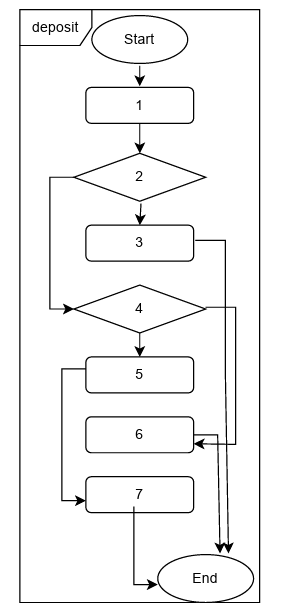
{ balance -= amount;5

**return** **true**;} }}6

**return** **false**;}7

|  |  |  |  |
| --- | --- | --- | --- |
| NO | Path | input | coverage |
| 1 | S 1,2,3,4,5,6 E | Amount=600  owner =Ahmed  balance=7000  password =123 |  |
| 2 | S 1,2,3,7 E | Amount=6000  owner =Ahmed  balance=7000  password =123 |  |
| 3 | S 1,2,7 E | Amount=768  owner =Mohammed  password =123 |  |
| 4 | S 1,2,3,4,7 E | Amount=4000  owner =Ali  balance=500  password =8989 |  |

Coverage=4\4=1\*100= 100%



**public** String deposit(**int** amount) {1

**if** (owner ==**null**){2

**return** **null**;}3

**if** (owner =="Ahmed" && password ==123){4

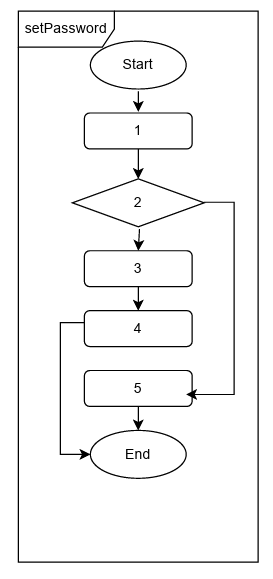
balance += amount;}5

**else** **return** "not successful";6

**return** "successful";}7

|  |  |  |  |
| --- | --- | --- | --- |
| NO | Path | input | coverage |
| 1 | S 1,2,4,5,7 E | Amount=7098  owner =Ahmed  password =123 |  |
| 2 | S 1,2,4,6 E | Amount=768  owner =Mohammed  password =123 |  |
| 3 | S 1,2,3 E | Amount=4000  owner =Ahmad  password =123 |  |
|  |  |  |  |

Coverage=2\3=0.66\*100= 66%



**public** **boolean** setPassword(**int** password) {1

**if**(99 <= password && password <= 99999999) {2

**this**.password = password; 3

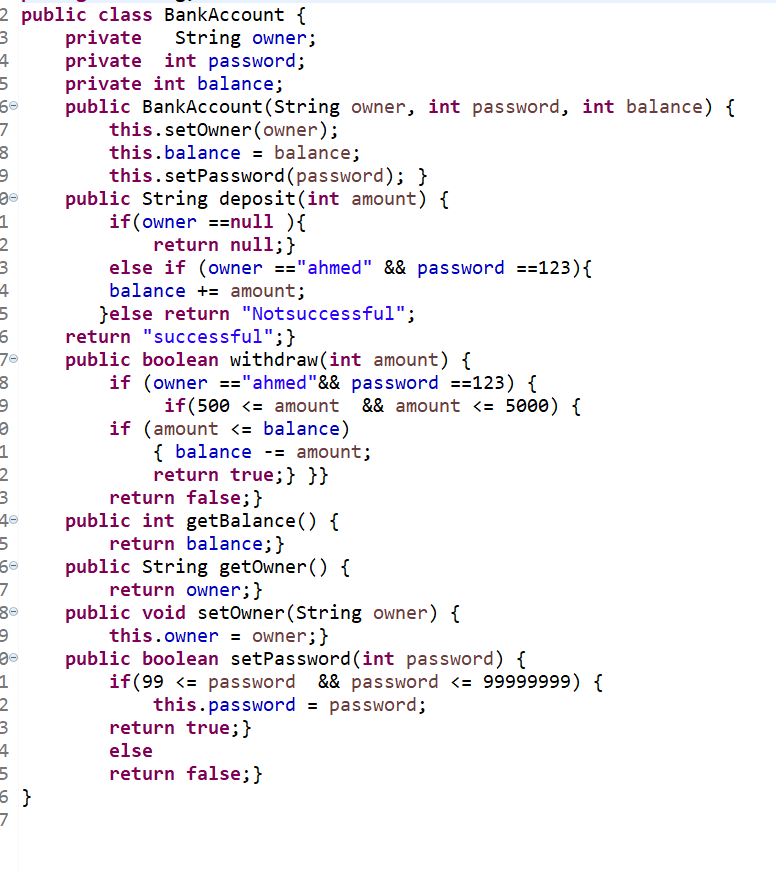
**return** **true**;} 4

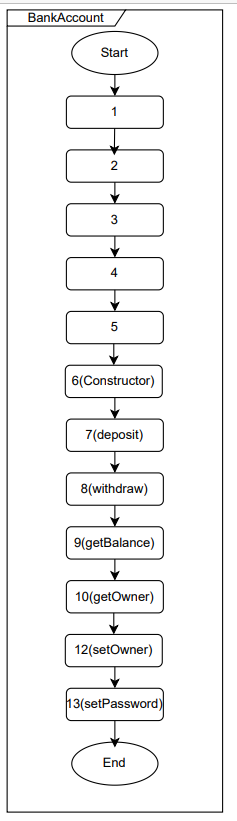
**else** **return** **false**;} }5

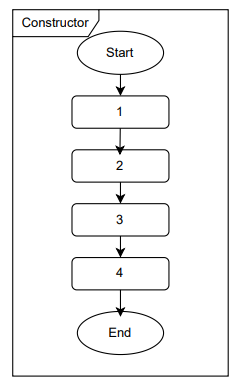
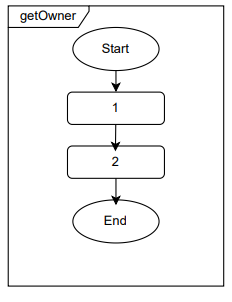
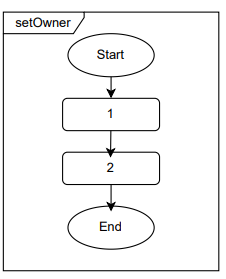
|  |  |  |  |
| --- | --- | --- | --- |
| NO | Path | input | coverage |
| 1 | S 1,2,3,4, E | password =123 |  |
| 3 | S 1,2,5 E | password =56987569756 |  |

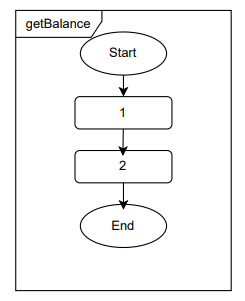
Coverage=2\2=1\*100= 100%

* **The full code with rest of flowcharts**

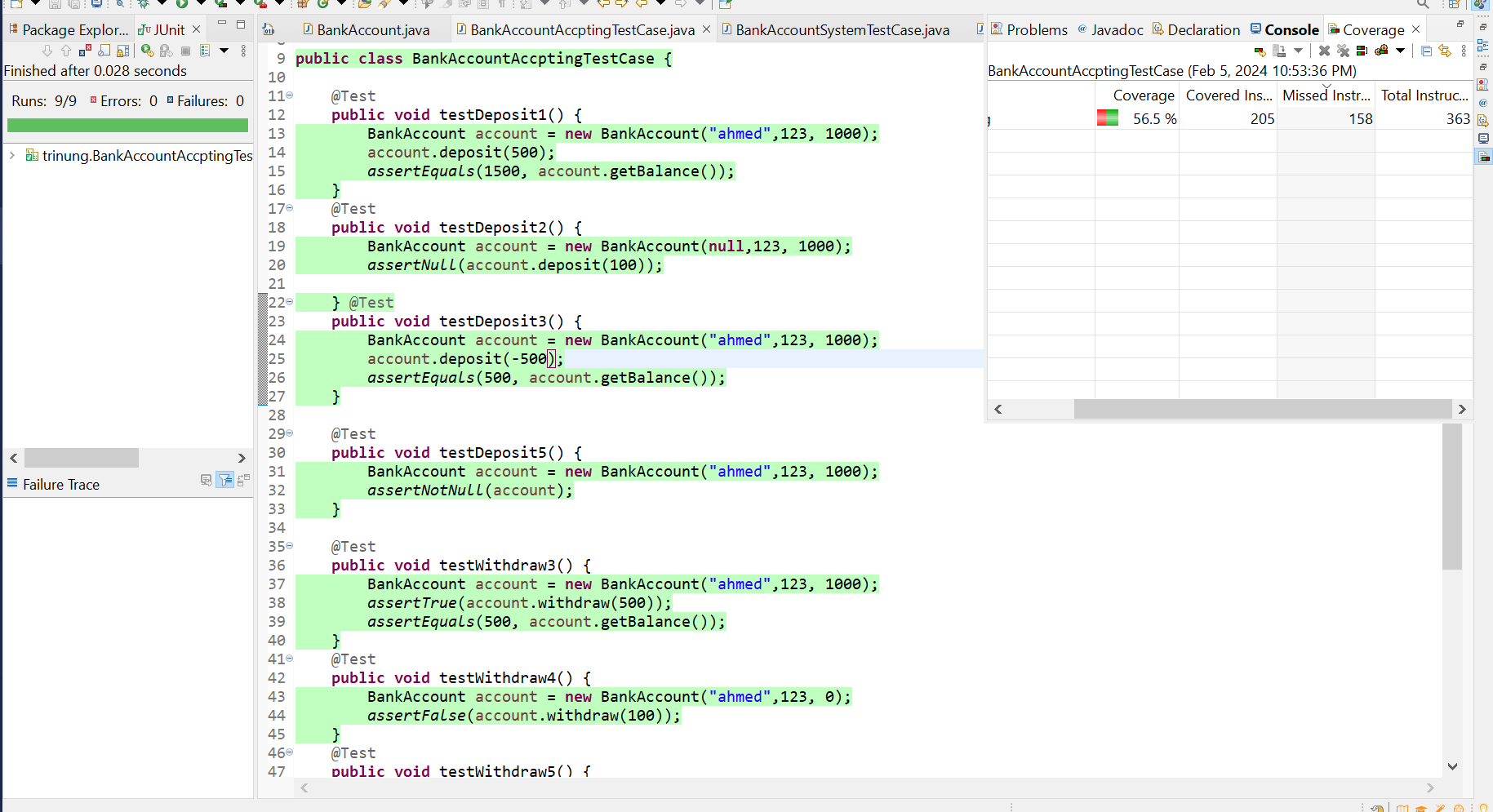


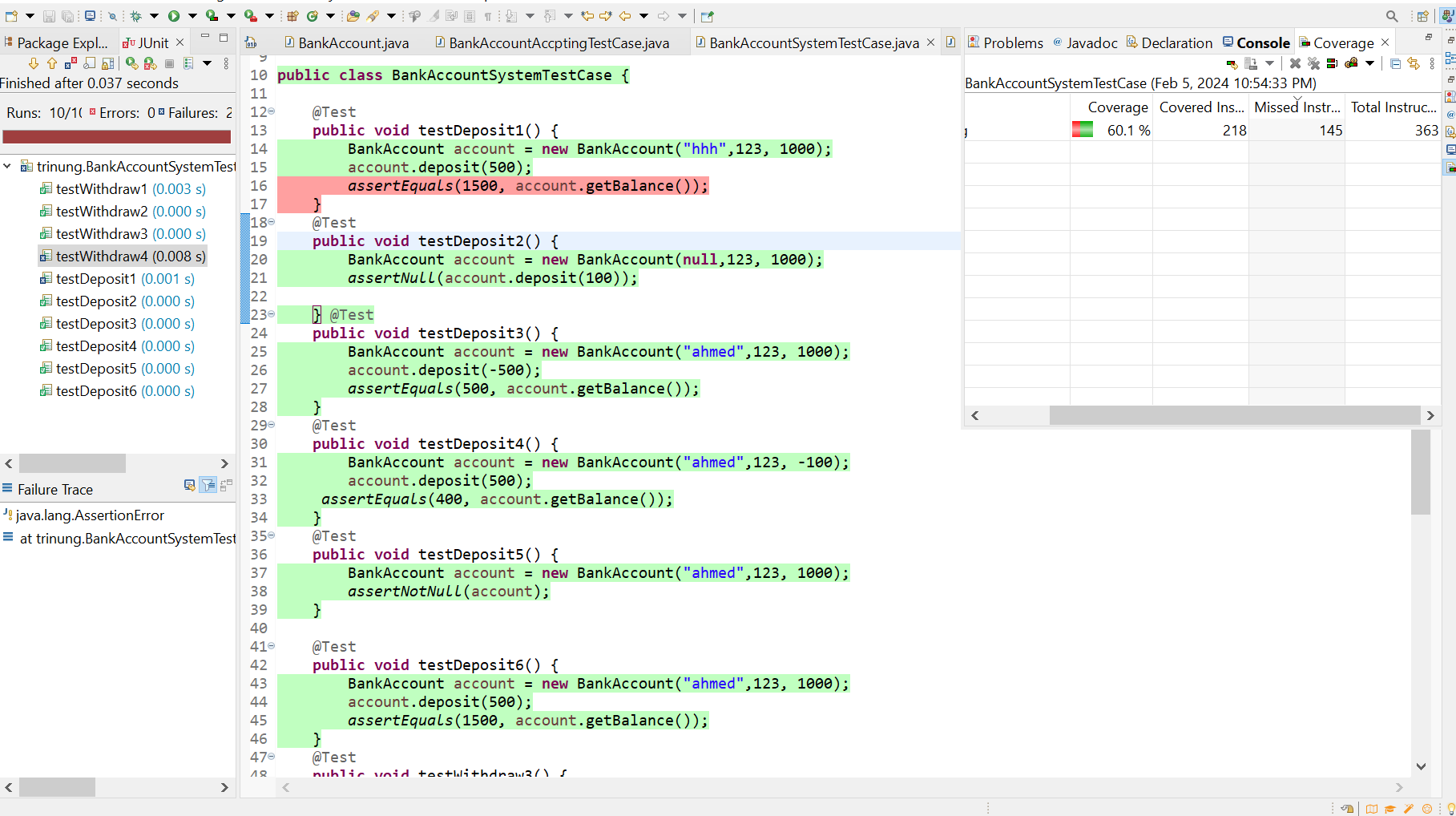






* 1. **Junit coverage**





**7\_ Uncovered Test**

**Operations that were not covered in the test**

|  |  |
| --- | --- |
| Operation | Reason |
| getOwner() | A tracking was done only because of the difficulty of the testing when there are no internal conditions and values |
| getBalance () | A tracking was done only because of the difficulty of the testing when there are no internal conditions and values |
| setOwner () | A tracking was done only because of the difficulty of the testing when there are no internal conditions and values |