

Lab4
Yuichi Hamamoto

Question 1:

The screenshot displays the IntelliJ IDEA test runner interface for two test runs. Each run shows a status bar at the top with icons for navigation and a summary of results. Below the status bar, a red progress bar indicates the test status. The test results are listed below the progress bar, showing the test class and the specific test method that failed.

Run 1:

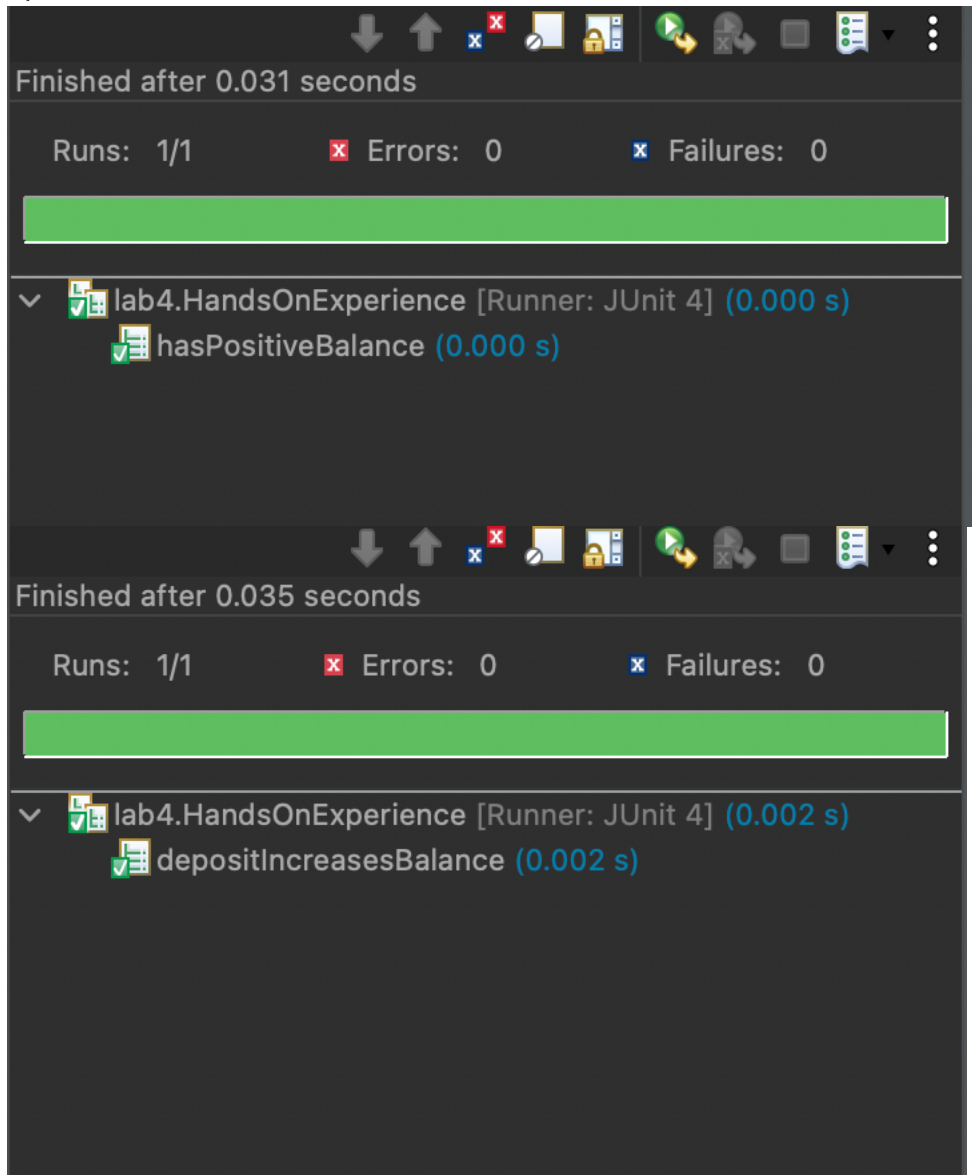
- Finished after 0.04 seconds
- Runs: 1/1 Errors: 1 Failures: 0
- lab4.HandsOnExperience [Runner: JUnit 4] (0.001 s)
- hasPositiveBalance (0.001 s)

Run 2:

- Finished after 0.025 seconds
- Runs: 1/1 Errors: 1 Failures: 0
- lab4.HandsOnExperience [Runner: JUnit 4] (0.003 s)
- depositIncreasesBalance (0.003 s)

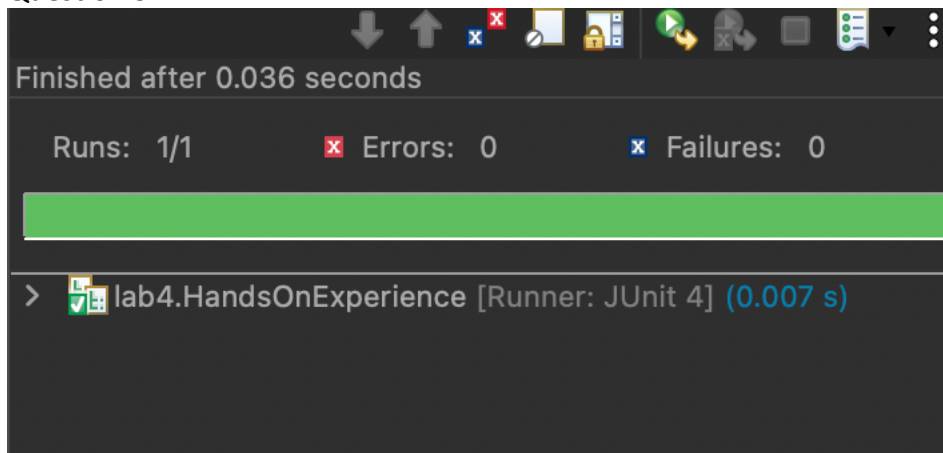
They both failed because “account” was not initialized.

Question 2:



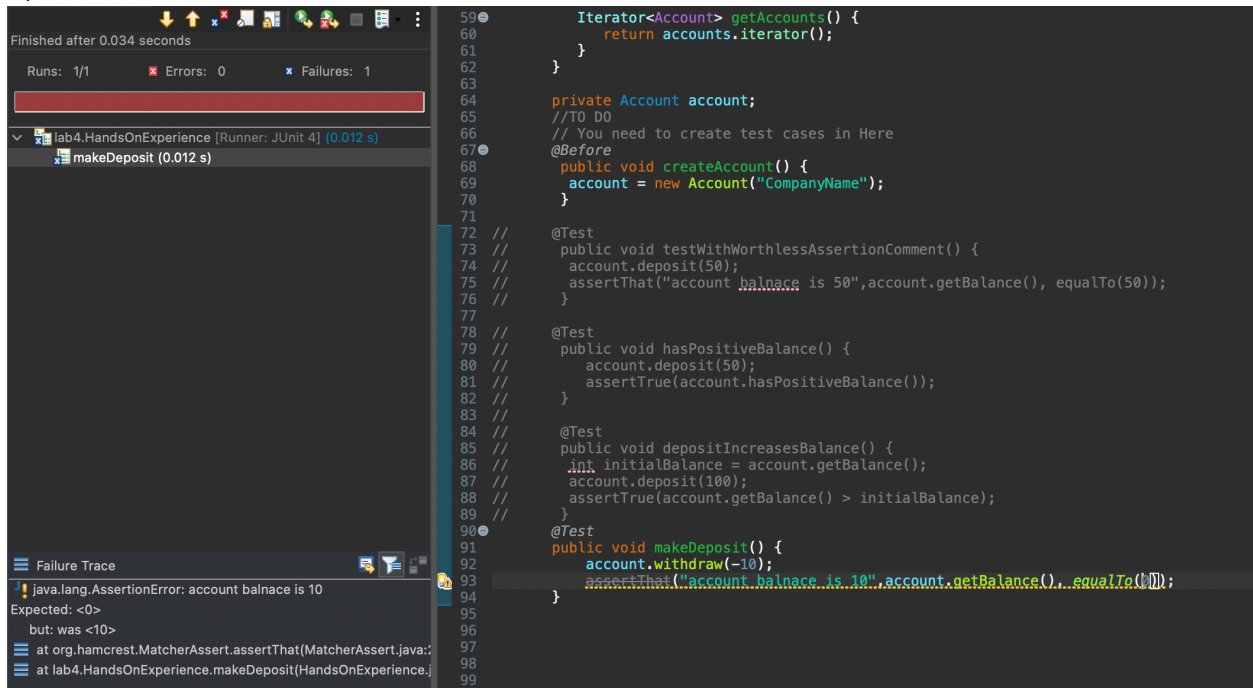
They successfully run because the previous problem is resolved. So, account is now initialized.

Question 3:



It runs successfully with comment. It'll show the comment when the test fails.

Question 4:



```
59  Iterator<Account> getAccounts() {
60      return accounts.iterator();
61  }
62
63
64  private Account account;
65  // TO DO
66  // You need to create test cases in Here
67  @Before
68  public void createAccount() {
69      account = new Account("CompanyName");
70  }
71
72  //
73  // @Test
74  // public void testWithWorthlessAssertionComment() {
75  //     account.deposit(50);
76  //     assertThat("account balance is 50", account.getBalance(), equalTo(50));
77  // }
78  //
79  // @Test
80  // public void hasPositiveBalance() {
81  //     account.deposit(50);
82  //     assertTrue(account.hasPositiveBalance());
83  // }
84  //
85  // @Test
86  // public void depositIncreasesBalance() {
87  //     int initialBalance = account.getBalance();
88  //     account.deposit(100);
89  //     assertTrue(account.getBalance() > initialBalance);
90  // }
91  @Test
92  public void makeDeposit() {
93      account.withdraw(-10);
94      assertThat("account balance is 10", account.getBalance(), equalTo(10));
95  }
96
97
98
99
```

Finished after 0.034 seconds

Runs: 1/1 Errors: 0 Failures: 1

lab4.HandsOnExperience [Runner: JUnit 4] (0.012 s)

makeDeposit (0.012 s)

Failure Trace

java.lang.AssertionError: account balance is 10
Expected: <0>
but: was <10>
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
at lab4.HandsOnExperience.makeDeposit(HandsOnExperience.java:94)

Expected result: 0

Because we should not be able to withdraw negative amount, the balance should remain as 0. However, because the function does not check if the parameter is positive it fails.

Question 5:

The screenshot displays two sequential test runs in an IDE. The top run shows a test named 'throwsWhenWithdrawingTooMuch' passing successfully, indicated by a green progress bar and a green checkmark icon. The bottom run shows the same test failing, indicated by a red progress bar and a red 'x' icon. The code on the right shows the test method being modified to use 'expected'.

Top Run Summary:

- Finished after 0.034 seconds
- Runs: 1/1
- Errors: 0
- Failures: 0

Test Results:

- lab4.HandsOnExperience [Runner: JUnit 4] (0.000 s)
- throwsWhenWithdrawingTooMuch (0.000 s)

Code Snippet (Top):

```
59 Iterator<Account> getAccounts() {  
60     return accounts.iterator();  
61 }  
62  
63  
64 private Account account;  
65 //T0 D0  
66 // You need to create test cases in Here  
67 @Before  
68 public void createAccount() {  
69     account = new Account("CompanyName");  
70 }  
71 @Test(expected = InsufficientFundsException.class)  
72 public void throwsWhenWithdrawingTooMuch() {  
73     account.withdraw(100);  
74 }  
75
```

Bottom Run Summary:

- Finished after 0.034 seconds
- Runs: 1/1
- Errors: 1
- Failures: 0

Test Results:

- lab4.HandsOnExperience [Runner: JUnit 4] (0.014 s)
- throwsWhenWithdrawingTooMuch (0.014 s)

Code Snippet (Bottom):

```
59 Iterator<Account> getAccounts() {  
60     return accounts.iterator();  
61 }  
62  
63  
64 private Account account;  
65 //T0 D0  
66 // You need to create test cases in Here  
67 @Before  
68 public void createAccount() {  
69     account = new Account("CompanyName");  
70 }  
71 @Test  
72 public void throwsWhenWithdrawingTooMuch() {  
73     account.withdraw(100);  
74 }  
75
```

With expected it passes and without it fails because with expect it checks if it throws exception or not.