Group: Joseph Stefanoni, Alice Zaytseva, Shrihit Saxena, and Hala Basyouni
SSW 555
April 24, 2025

Assignment 3 - Practice Refactoring

User Story:

- **Main User Story: Bank System**
  - As a user, I want to manage my bank account (deposit, withdraw, transfer funds, and check my balance) so that I can securely control my finances.
- **Detailed User Stories:**
  1. **Deposit Money:** As a user, I want to deposit money into my account so that I can increase my balance.
  2. **Withdraw Money:** As a user, I want to withdraw money from my account so that I can access my funds.
  3. **Transfer Money:** As a user, I want to transfer money from my account to another user's account so that I can send money easily.
  4. **Prevent Overdraft:** As a user, I want to be prevented from withdrawing or transferring more money than I have so that I don't go into debt.
  5. **View Balance:** As a user, I want to view my account balance so that I can see how much money I have.

Manual Test Cases (Gherkin Language):

1. **Feature:** Deposit money into checking account
   - **Scenario:** User deposits money successfully
     - Given a user with a checking account balance of $0
     - When the user deposits $100 into the checking account
     - Then the checking account balance should be $100

2. **Feature:** Deposit money into savings account
   - **Scenario:** User deposits money successfully
     - Given a user with a savings account balance of $0
     - When the user deposits $200 into the savings account
     - Then the savings account balance should be $200

3. **Feature:** Withdraw money from checking account
   - **Scenario:** User withdraws money successfully
     - Given a user with a checking account balance of $150
     - When the user withdraws $50 from the checking account
     - Then the checking account balance should be $100

4. **Feature:** Withdraw money from savings account
   - **Scenario:** User withdraws money successfully
     - Given a user with a savings account balance of $300
     - When the user withdraws $100 from the savings account
     - Then the savings account balance should be $200

5. **Feature:** Transfer money from checking to another account
   - **Scenario:** User transfers money successfully
     - Given a user A with a checking account balance of $500
     - And a user B with a checking account balance of $0
     - When user A transfers $200 to user B
     - Then user A's checking balance should be $300
     - And user B's checking balance should be $200

6. **Feature:** View account balances
   - **Scenario:** User views their checking and savings balances
     - Given a user with a checking balance of $120 and savings balance of $380
     - When the user requests to view their balances
     - Then the system should display $120 for checking and $380 for savings

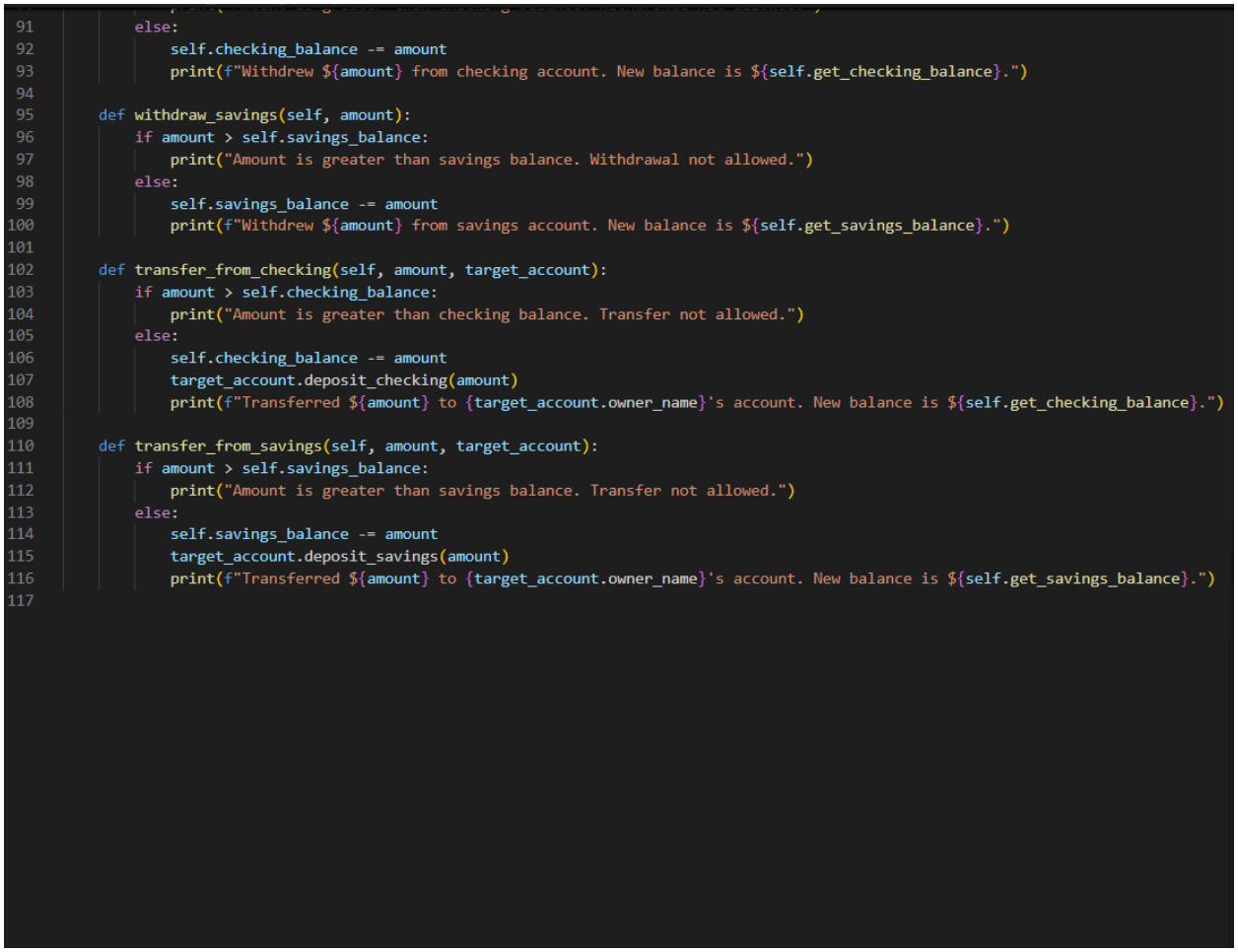
Bad Smells:

In our initial code, we identified a few bad smells. The first two are duplicated code and unnecessary complexity by having the similar, but separate methods for the savings account and checking account specifically. Because of this, the class itself is fairly large, which is another bad smell that requires refactoring. Even though this code works as is, refactoring could help make the code easier to read and reduce complexity.

Test Results Picture:

C: > Users > Shrihit > Downloads > bank_system.py > ...

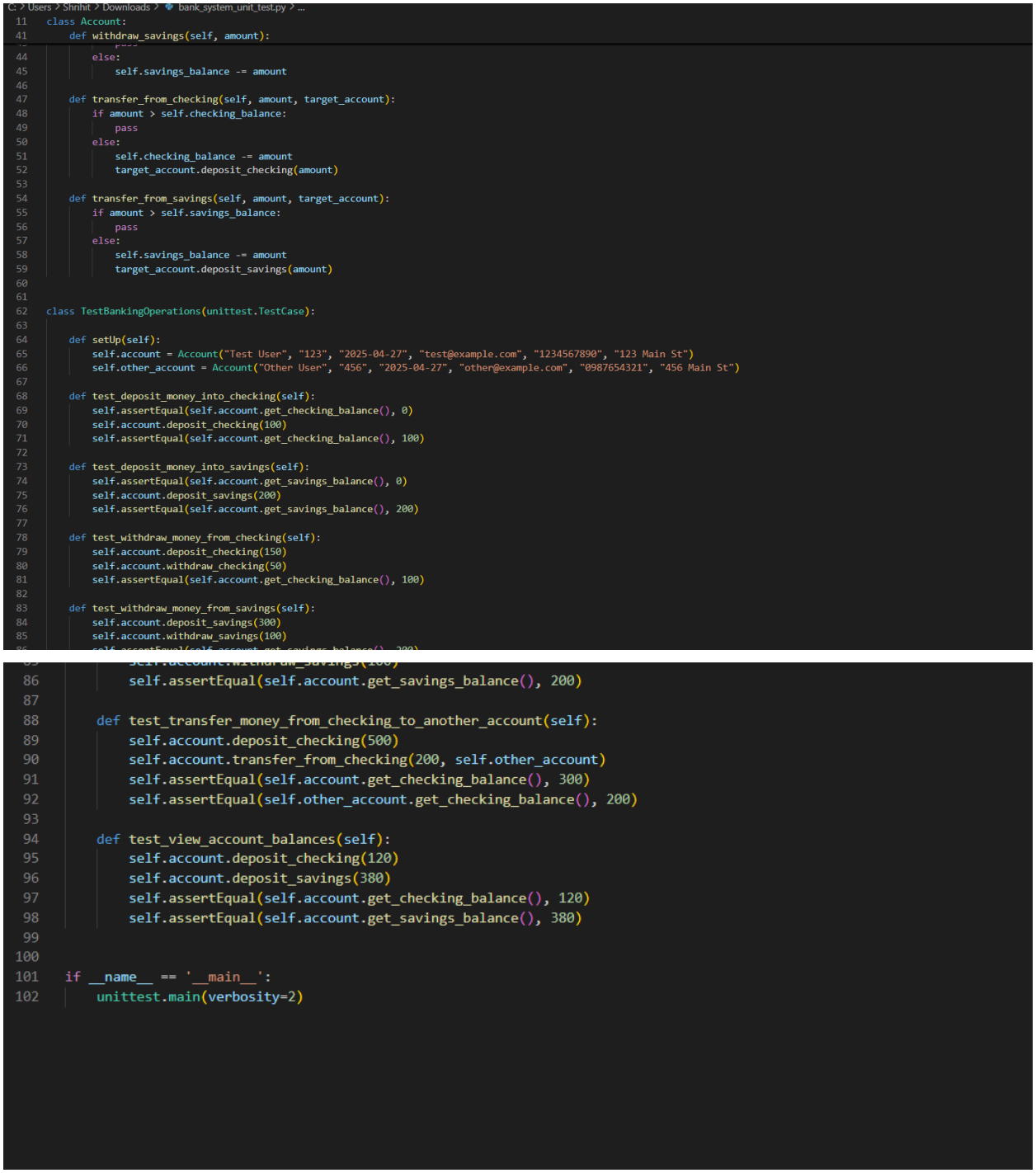
```
"""
    Assignment 3 - Practice Refactoring
    Group: Joseph Stefanoni, Hala Basyouni, Shrihit Saxena, Alice Zaytseva
    "I pledge my honor that I have abided by the Stevens Honor System."
    April 27, 2025

    Main User Story: Bank System
        As a user, I want to manage my bank account (deposit, withdraw, transfer funds, and check my balance) so that I can securely control my finances.
    Detailed User Stories:
        1. Deposit Money: As a user, I want to deposit money into my account so that I can increase my balance.
        2. Withdraw Money: As a user, I want to withdraw money from my account so that I can access my funds.
        3. Transfer Money: As a user, I want to transfer money from my account to another user's account so that I can send money easily.
        4. Prevent Overdraft: As a user, I want to be prevented from withdrawing or transferring more money than I have so that I don't go into debt.
        5. View Balance: As a user, I want to view my account balance so that I can see how much money I have.

    Manual Test Cases (Gherkin Language):
        1. Feature: Deposit money into checking account
            - Scenario: User deposits money successfully
                - Given a user with a checking account balance of $0
                - When the user deposits $100 into the checking account
                - Then the checking account balance should be $100

        2. Feature: Deposit money into savings account
            - Scenario: User deposits money successfully
                - Given a user with a savings account balance of $0
                - When the user deposits $200 into the savings account
                - Then the savings account balance should be $200

        3. Feature: Withdraw money from checking account
            - Scenario: User withdraws money successfully
                - Given a user with a checking account balance of $150
                - When the user withdraws $50 from the checking account
                - Then the checking account balance should be $100

        4. Feature: Withdraw money from savings account
            - Scenario: User withdraws money successfully
                - Given a user with a savings account balance of $300
                - When the user withdraws $100 from the savings account
                - Then the savings account balance should be $200

        5. Feature: Transfer money from checking to another account
            - Scenario: User transfers money successfully
                - Given a user A with a checking account balance of $500
                - And a user B with a checking account balance of $0
                - When user A transfers $200 to user B
                - Then user A's checking balance should be $300
                - And user B's checking balance should be $200

        6. Feature: View account balances
            - Scenario: User views their checking and savings balances
                - Given a user with a checking balance of $120 and savings balance of $380
                - When the user requests to view their balances
                - Then the system should display $120 for checking and $380 for savings
"""

class Account:

    def __init__(self, owner_name, account_id, date_created, email, phone_number, address):
        self.owner_name = owner_name
        self.account_id = account_id
        self.date_created = date_created
        self.checking_balance = 0.0
        self.savings_balance = 0.0
        self.email = email
        self.phone_number = phone_number
        self.address = address

    def deposit_checking(self, amount):
        self.checking_balance += amount
        print(f"Deposited ${amount} to checking account. New balance is ${self.get_checking_balance}.")

    def deposit_savings(self, amount):
        self.savings_balance += amount
        print(f"Deposited ${amount} to savings account. New balance is ${self.get_savings_balance}.")

    def get_checking_balance(self):
        return self.checking_balance

    def print_checking_balance(self):
        print(f"Checking account balance: ${self.get_checking_balance()}")

    def get_savings_balance(self):
        return self.savings_balance

    def print_savings_balance(self):
        print(f"Savings account balance: ${self.get_savings_balance()}")

    def withdraw_checking(self, amount):
        if amount > self.checking_balance:
            print("Amount is greater than checking balance. Withdrawal not allowed.")
```



```
91          else:
92              self.checking_balance -= amount
93              print(f"Withdrew ${amount} from checking account. New balance is ${self.get_checking_balance}.")
94
95      def withdraw_savings(self, amount):
96          if amount > self.savings_balance:
97              print("Amount is greater than savings balance. Withdrawal not allowed.")
98          else:
99              self.savings_balance -= amount
100             print(f"Withdrew ${amount} from savings account. New balance is ${self.get_savings_balance}.")
101
102     def transfer_from_checking(self, amount, target_account):
103         if amount > self.checking_balance:
104             print("Amount is greater than checking balance. Transfer not allowed.")
105         else:
106             self.checking_balance -= amount
107             target_account.deposit_checking(amount)
108             print(f"Transferred ${amount} to {target_account.owner_name}'s account. New balance is ${self.get_checking_balance}.")
109
110     def transfer_from_savings(self, amount, target_account):
111         if amount > self.savings_balance:
112             print("Amount is greater than savings balance. Transfer not allowed.")
113         else:
114             self.savings_balance -= amount
115             target_account.deposit_savings(amount)
116             print(f"Transferred ${amount} to {target_account.owner_name}'s account. New balance is ${self.get_savings_balance}.")
117
```

Initial Code

```python
"""
    Assignment 3 - Practice Refactoring
    Group: Joseph Stefanoni, Hala Basyouni, Shrihit Saxena, Alice Zaytseva
    "I pledge my honor that I have abided by the Stevens Honor System."
    April 27, 2025

    Description: Unit tests for the Bank System user story.
"""
import unittest

class Account:

    def __init__(self, owner_name, account_id, date_created, email, phone_number, address):
        self.owner_name = owner_name
        self.account_id = account_id
        self.date_created = date_created
        self.checking_balance = 0.0
        self.savings_balance = 0.0
        self.email = email
        self.phone_number = phone_number
        self.address = address

    def deposit_checking(self, amount):
        self.checking_balance += amount

    def deposit_savings(self, amount):
        self.savings_balance += amount

    def get_checking_balance(self):
        return self.checking_balance

    def get_savings_balance(self):
        return self.savings_balance

    def withdraw_checking(self, amount):
        if amount > self.checking_balance:
            pass
        else:
            self.checking_balance -= amount

    def withdraw_savings(self, amount):
        if amount > self.savings_balance:
            pass
        else:
            self.savings_balance -= amount
```



```
C: > Users > Shrihit > Downloads > bank_system_unit_test.py > ...
class Account:
    def withdraw_savings(self, amount):
        else:
            self.savings_balance -= amount

    def transfer_from_checking(self, amount, target_account):
        if amount > self.checking_balance:
            pass
        else:
            self.checking_balance -= amount
            target_account.deposit_checking(amount)

    def transfer_from_savings(self, amount, target_account):
        if amount > self.savings_balance:
            pass
        else:
            self.savings_balance -= amount
            target_account.deposit_savings(amount)


class TestBankingOperations(unittest.TestCase):

    def setUp(self):
        self.account = Account("Test User", "123", "2025-04-27", "test@example.com", "1234567890", "123 Main St")
        self.other_account = Account("Other User", "456", "2025-04-27", "other@example.com", "0987654321", "456 Main St")

    def test_deposit_money_into_checking(self):
        self.assertEqual(self.account.get_checking_balance(), 0)
        self.account.deposit_checking(100)
        self.assertEqual(self.account.get_checking_balance(), 100)

    def test_deposit_money_into_savings(self):
        self.assertEqual(self.account.get_savings_balance(), 0)
        self.account.deposit_savings(200)
        self.assertEqual(self.account.get_savings_balance(), 200)

    def test_withdraw_money_from_checking(self):
        self.account.deposit_checking(150)
        self.account.withdraw_checking(50)
        self.assertEqual(self.account.get_checking_balance(), 100)

    def test_withdraw_money_from_savings(self):
        self.account.deposit_savings(300)
        self.account.withdraw_savings(100)
        self.assertEqual(self.account.get_savings_balance(), 200)

    def test_transfer_money_from_checking_to_another_account(self):
        self.account.deposit_checking(500)
        self.account.transfer_from_checking(200, self.other_account)
        self.assertEqual(self.account.get_checking_balance(), 300)
        self.assertEqual(self.other_account.get_checking_balance(), 200)

    def test_view_account_balances(self):
        self.account.deposit_checking(120)
        self.account.deposit_savings(380)
        self.assertEqual(self.account.get_checking_balance(), 120)
        self.assertEqual(self.account.get_savings_balance(), 380)


if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Tester Code

SSW-555 / assignment_3-Practice_Refactoring / **first_test_output.txt**

aZayst Test cases and output added — 05b34f7 · 15 hours ago — History

Code | Blame | 13 lines (10 loc) · 857 Bytes | Raw

```
alice@Alices-MacBook-Pro assignment_3-Practice_Refactoring % python3 bank_system_unit_test.py
test_deposit_money_into_checking (__main__.TestBankingOperations.test_deposit_money_into_checking) ... ok
test_deposit_money_into_savings (__main__.TestBankingOperations.test_deposit_money_into_savings) ... ok
test_transfer_money_from_checking_to_another_account (__main__.TestBankingOperations.test_transfer_money_from_checking_to_another_account) ... ok
test_view_account_balances (__main__.TestBankingOperations.test_view_account_balances) ... ok
test_withdraw_money_from_checking (__main__.TestBankingOperations.test_withdraw_money_from_checking) ... ok
test_withdraw_money_from_savings (__main__.TestBankingOperations.test_withdraw_money_from_savings) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK
```

Test Output

New Test Results Picture:

refactored_bank_system.py > Account

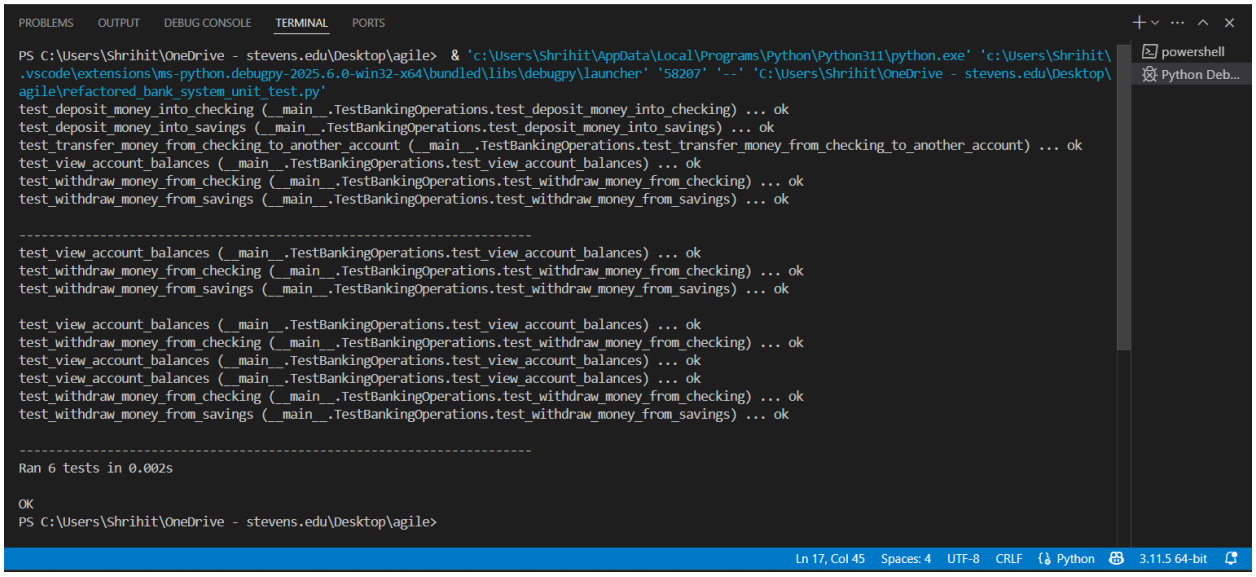
```python
class Account:
    def __init__(self, owner_name, account_id, date_created, email, phone_number, address):
        self.owner_name = owner_name
        self.account_id = account_id
        self.date_created = date_created
        self.balances = {
            'checking': 0.0,
            'savings': 0.0
        }
        self.email = email
        self.phone_number = phone_number
        self.address = address

    def deposit(self, account_type, amount):
        if account_type in self.balances:
            self.balances[account_type] += amount
            return f"Deposited ${amount} to {account_type}."
        else:
            return "Invalid account type."

    def withdraw(self, account_type, amount):
        if account_type in self.balances:
            if amount <= self.balances[account_type]:
                self.balances[account_type] -= amount
                return f"Withdrew ${amount} from {account_type}."
            else:
                return f"Insufficient funds in {account_type}."
        else:
            return "Invalid account type."

    def transfer(self, from_account_type, amount, target_account):
        if from_account_type in self.balances:
            if amount <= self.balances[from_account_type]:
                self.balances[from_account_type] -= amount
                target_account.deposit(from_account_type, amount)
                return f"Transferred ${amount} from {from_account_type} to {target_account.owner_name}."
            else:
                return f"Insufficient funds to transfer from {from_account_type}."
        else:
            return "Invalid account type."

    def get_balance(self, account_type):
        return self.balances.get(account_type, "Invalid account type.")

    def view_balances(self):
        return f"Checking: ${self.balances['checking']}, Savings: ${self.balances['savings']}"

```

Refactored code

refactored_bank_system_unit_test.py > TestBankingOperations > test_deposit_money_into_checking

```python
import unittest
from refactored_bank_system import Account

class TestBankingOperations(unittest.TestCase):

    def setUp(self):
        self.account = Account("Test1", "123", "2025-04-27", "test1@example.com", "1234567890", "123 Main St")
        self.other_account = Account("Test2", "456", "2025-04-27", "test2@example.com", "0987654321", "456 Main St")

    def test_deposit_money_into_checking(self):
        self.assertEqual(self.account.get_balance('checking'), 0)
        self.account.deposit('checking', 100)
        self.assertEqual(self.account.get_balance('checking'), 100)

    def test_deposit_money_into_savings(self):
        self.assertEqual(self.account.get_balance('savings'), 0)
        self.account.deposit('savings', 200)
        self.assertEqual(self.account.get_balance('savings'), 200)

    def test_withdraw_money_from_checking(self):
        self.account.deposit('checking', 150)
        self.account.withdraw('checking', 50)
        self.assertEqual(self.account.get_balance('checking'), 100)

    def test_withdraw_money_from_savings(self):
        self.account.deposit('savings', 300)
        self.account.withdraw('savings', 100)
        self.assertEqual(self.account.get_balance('savings'), 200)

    def test_transfer_money_from_checking_to_another_account(self):
        self.account.deposit('checking', 500)
        self.account.transfer('checking', 200, self.other_account)
        self.assertEqual(self.account.get_balance('checking'), 300)
        self.assertEqual(self.other_account.get_balance('checking'), 200)

    def test_view_account_balances(self):
        self.account.deposit('checking', 120)
        self.account.deposit('savings', 380)
        self.assertEqual(self.account.get_balance('checking'), 120)
        self.assertEqual(self.account.get_balance('savings'), 380)

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Refactored Test Code



Refactored Test Output