**Exercise: Enhanced Bank Account Management with File Operations**

Create a Python program that simulates an enhanced bank account management system with the capability to save and load account data to/from a text file. Implement a class named **BankAccount** with the following features:

1. The class should have attributes:
   * **account\_number** (string): A unique account number.(if the account exists, print message to user that account already exists)
   * **account\_holder** (string): The name of the account holder.
   * **balance** (float): The current balance of the account.
2. Implement methods:
   * **\_\_init\_\_(self, account\_number, account\_holder, initial\_balance)**: Initializes a new bank account with the provided account number, account holder name, and initial balance.
   * **deposit(self, amount)**: Adds the specified amount to the account balance. Ensure that the deposit amount is positive.
   * **withdraw(self, amount)**: Subtracts the specified amount from the account balance. Ensure that the withdrawal amount is positive and does not result in a negative balance.
   * **get\_balance(self)**: Returns the current balance of the account.
   * **transfer(self, target\_account, amount)**: Transfers funds from the current account to a target account. Ensure that the transfer amount is positive, and the current account has sufficient funds.
   * **save\_to\_file(self, filename="bank\_accounts.txt")**: Saves the account data to a text file. Each line in the file should represent an account with the format **account\_number,account\_holder,balance**.
   * **load\_from\_file(filename="bank\_accounts.txt")**: Loads account data from a text file and returns a list of **BankAccount** instances.
3. Create a separate function named **update\_account\_data(account1, account2)** that demonstrates how to use the **save\_to\_file** and **load\_from\_file** methods to update the data in the file. The function should save the account data to a file, load it back, and display the loaded account data for verification.

**Example Usage:**

# Create Bank Accounts

account1 = BankAccount("123456", "Alice", 1000.0)

account2 = BankAccount("789012", "Bob", 500.0)

# Perform Transactions

account1.deposit(500.0)

account1.withdraw(200.0)

account2.deposit(300.0)

account1.transfer(account2, 150.0)

# Display Current Balances

print(f"\nCurrent Balance for {account1.account\_holder}'s account: ${account1.get\_balance()}")

print(f"Current Balance for {account2.account\_holder}'s account: ${account2.get\_balance()}")

# Update Account Data in a File

update\_account\_data(account1, account2)

**Solution:**

import os

class BankAccount:

def \_\_init\_\_(self, account\_number, account\_holder, initial\_balance):

"""

Initializes a new bank account with the provided details.

Parameters:

- account\_number (str): A unique account number.

- account\_holder (str): The name of the account holder.

- initial\_balance (float): The initial balance of the account.

"""

self.account\_number = account\_number

self.account\_holder = account\_holder

self.balance = initial\_balance

def deposit(self, amount):

"""

Adds the specified amount to the account balance.

Parameters:

- amount (float): The amount to be deposited.

Ensures that the deposit amount is positive.

"""

if amount > 0:

self.balance += amount

print(f"Deposited ${amount}. New Balance: ${self.balance}")

else:

print("Invalid deposit amount. Please enter a positive value.")

def withdraw(self, amount):

"""

Subtracts the specified amount from the account balance.

Parameters:

- amount (float): The amount to be withdrawn.

Ensures that the withdrawal amount is positive and does not result in a negative balance.

"""

if 0 < amount <= self.balance:

self.balance -= amount

print(f"Withdrew ${amount}. New Balance: ${self.balance}")

elif amount > self.balance:

print("Insufficient funds for withdrawal.")

else:

print("Invalid withdrawal amount. Please enter a positive value.")

def get\_balance(self):

"""

Returns the current balance of the account.

"""

return self.balance

def transfer(self, target\_account, amount):

"""

Transfers funds from the current account to a target account.

Parameters:

- target\_account (BankAccount): The target account to transfer funds to.

- amount (float): The amount to be transferred.

Ensures that the transfer amount is positive and the current account has sufficient funds.

"""

if 0 < amount <= self.balance:

self.balance -= amount

target\_account.deposit(amount)

print(f"Transferred ${amount} to {target\_account.account\_holder}. New Balance: ${self.balance}")

elif amount > self.balance:

print("Insufficient funds for transfer.")

else:

print("Invalid transfer amount. Please enter a positive value.")

def save\_to\_file(self, filename="bank\_accounts.txt"):

"""

Saves the account data to a text file.

Parameters:

- filename (str): The name of the file to save the data to. Default is "bank\_accounts.txt".

Each line in the file represents an account with the format account\_number,account\_holder,balance.

"""

with open(filename, 'a') as file:

file.write(f"{self.account\_number},{self.account\_holder},{self.balance}\n")

@staticmethod

def load\_from\_file(filename="bank\_accounts.txt"):

"""

Loads account data from a text file and returns a list of BankAccount instances.

Parameters:

- filename (str): The name of the file to load the data from. Default is "bank\_accounts.txt".

Returns:

- List[BankAccount]: A list of BankAccount instances loaded from the file.

"""

accounts = []

if os.path.exists(filename):

with open(filename, 'r') as file:

for line in file:

account\_data = line.strip().split(',')

if len(account\_data) == 3:

account = BankAccount(account\_data[0], account\_data[1], float(account\_data[2]))

accounts.append(account)

return accounts

# New Function to Demonstrate File Operations

def update\_account\_data(account1, account2):

"""

Demonstrates saving account data to a file, loading it back, and displaying the loaded account data.

Parameters:

- account1 (BankAccount): The first bank account.

- account2 (BankAccount): The second bank account.

"""

# Save account data to a file

account1.save\_to\_file()

account2.save\_to\_file()

# Load account data from the file

loaded\_accounts = BankAccount.load\_from\_file()

# Display loaded account data

print("\nLoaded Account Data:")

for loaded\_account in loaded\_accounts:

print(f"Account Number: {loaded\_account.account\_number}, Account Holder: {loaded\_account.account\_holder}, Balance: ${loaded\_account.balance}")

# Example Usage:

account1 = BankAccount("123456", "Alice", 1000.0)

account2 = BankAccount("789012", "Bob", 500.0)

# Perform Transactions

account1.deposit(500.0)

account1.withdraw(200.0)

account2.deposit(300.0)

account1.transfer(account2, 150.0)

# Display Current Balances

print(f"\nCurrent Balance for {account1.account\_holder}'s account: ${account1.get\_balance()}")

print(f"Current Balance for {account2.account\_holder}'s account: ${account2.get\_balance()}")

# Update Account Data in a File

update\_account\_data(account1, account2)