CSCI 3060U - Software Quality and Assurance

Project Phase II - Class and Method Description Table

Wenbo Zhang, Xuan Zheng, Neel samirkumar Shah, Dev rajivkumar Thaker

Description

This program simulates a banking system that allows users to perform transactions such as withdrawals, transfers, and bill payments. Users must log in before performing any transaction. The system supports both standard (user) and admin sessions, where admin sessions have special privileges (e.g., overriding account limits, accessing all user accounts).

Input Sources:
- No dedicated input file for accounts (hardcoded users)
- CLI-based user input for commands

Output File:
- daily_transaction_file.txt: Stores a record of all transactions performed during a session.

How to Run:
1. Execute main.py to start the application.
2. Enter "login" to authenticate:
- Enter session type ("admin" or "standard").
- Admin users get full access; standard users enter their account number.
3. After successful login, you can enter one of the following commands:
- "withdrawal" to withdraw funds.
- "transfer" to move funds between accounts.
- "paybill" to pay a bill to a known company.
- "create" (admin only) to create a new account.
- "delete" (admin only) to remove an existing account.
- "disable" (admin only) to disable an account.
- "changeplan" (admin only) to switch or toggle the account’s plan.
- "logout" to end the session and save all transactions.
4. Transaction details are automatically written to daily_transaction_file.txt upon each operation.

Class Description Table

Classes	Intentions
User (in main.py)	Represents a bank account holder with fields for account_number, user_name, availability (active/disabled), balance, and a default user_type. Primarily used for storing and updating account data.
Check (in check.py)	A utility/validation class providing various checks (e.g., existence, balance, amount limits, availability, etc.) used by all transaction classes.
Login (in login.py)	Handles user login logic. Checks session type (admin or standard) and validates the user to set a logged-in state.
Logout (in logout.py)	Manages the session logout process. Validates whether a user is logged in and finalizes the session by generating a logout transaction line.
Withdrawal (in withdrawal.py)	Handles withdrawing funds from an active user account. Validates account existence and balance before deducting the amount.
Transfer (in transfer.py)	Transfers funds between two accounts. Validates source and destination accounts, plus additional checks like availability and transfer limits.
Paybill (in paybill.py)	Pays a bill to a recognized company from the user's account. Validates company code, bill payment limits, and account balance.
Deposit (in deposit.py)	Deposits a specified amount into a user account. Validates the deposit amount and the account's active status before updating the balance.
Create (in create.py)	Creates a new bank account (privileged transaction for admins). Generates a unique account number, sets initial balance, and logs the creation transaction.
Delete (in delete.py)	Deletes an existing bank account (admin only). Validates the account holder name and account number, then logs a deletion transaction.
ChangePlan (in changeplan.py)	Changes a user's transaction plan (e.g., Student Plan to Non-Student Plan). Validates admin privileges, correct account number, and availability.

Disable (in disable.py)	Disables (sets to inactive) an existing account, preventing further transactions. Validates admin mode and ensures the account is not already disabled.

Methods Description Table

Class	Method	Description
User (in main.py)	No special methods	The constructor __init__ sets account details. Used primarily by transaction classes to access/update balances.
Check (in check.py)	user_check(user1, user2)	Ensures that two user accounts are not the same.
	availability_check(user)	Returns True if a user's account is active ("A"), otherwise False.
	balance_check(user, amount)	Checks if user.balance >= amount.
	limit_check(amount, limit)	Checks if the amount does not exceed the given limit.
	negative_amount_check(amount)	Returns True if amount > 0.
	zero_amount_check(amount)	Ensures the amount is not zero (amount != 0).
	account_existence_check(user)	Returns True if user is not None.
	admin_override_check(user)	Checks if user.userType == "admin".
	valid_company_check(company)	Validates whether company is recognized (EC, CQ, FI).
	company_id_check(company)	Returns a company ID (e.g., "10000") for the provided company.
	invalid_character_check(value)	Tries converting value to float. Returns True if numeric.
	missing_input_check(**kwargs)	Ensures no fields in kwargs are missing or empty.
	sender_account_match(user, sender_account)	Verifies that user.account_number == sender_account.
Login (in	__init__(userType, user,	Constructor: stores session type and user

login.py)	logged_in)	reference.
	check_user_type()	Ensures the session type is either "admin" or "standard".
	check_username()	Uses Check.account_existence_check to validate the user object is not None.
	check_double_login()	Ensures the user is not already logged in.
	process_login()	Orchestrates user login checks and prints success message.
	return_transaction_output()	Returns a formatted login transaction string (e.g., "01_{username}_").
Logout (in logout.py)	__init__(logged_in, session_type, current_user)	Constructor: sets the current session state and a Check object.
	process_logout()	Verifies if session is active, ensures not already processed, and finalizes logout. Returns True on success.
	return_transaction_output()	Returns a fixed string "00_____00000_00000.00__" for logging the logout transaction.
Withdrawal (in withdrawal.py)	__init__(user, amount)	Constructor: references a User object and an amount to withdraw.
	check_account_number()	Validates user existence via Check.account_existence_check.
	check_balance()	Checks if user's balance is sufficient.
	process_withdrawal()	Runs a series of checks (missing input, numeric, positive, etc.) and then deducts from user.balance.
	return_transaction_output()	Returns a formatted transaction line (02 or 03 code, depending on your code duplication).
Transfer (in transfer.py)	__init__(userType, user1, user2, amount, limit=1000.00)	Constructor: references source user (user1), destination (user2), session type, limit, etc.
	process_transfer()	Validates everything (existence, availability, numeric, limit checks). Updates balances if okay.
	display_transaction_output()	Prints the formatted transaction line (transfer code "02").

	return_transaction_output()	Returns the same formatted string for logging to file.
Paybill (in paybill.py)	__init__(userType, user, company, amount=None, limit=2000.00)	Constructor: sets user/session details for paying a bill.
	process_paybill()	Checks all required conditions (company code, limit, funds) and updates user's balance on success.
	company_check()	Internal method checking if company is known in COMPANY_ACCOUNTS.
	display_transaction_output(company_id)	Prints formatted transaction line with code "03" plus an end-session code.
	return_transaction_output(company_id)	Returns the transaction string for logging.
Deposit (in deposit.py)	__init__(userType, user, amount=None)	Constructor: references the user, deposit amount, and session type.
	process_deposit()	Validates deposit input (numeric, positive, non-zero, active account) and updates user's balance.
	return_transaction_output()	Returns the deposit transaction line (e.g., "04_{username}_{account_number}_{amount}").
Create (in create.py)	__init__(userType, accounts, transaction_file="daily_transaction_file.txt")	Constructor: sets userType, references the accounts dict, sets default transaction file.
	process_creation()	Admin-only. Prompts for account holder name and initial balance, creates a new account, logs transaction.
	return_transaction_output(new_account, initial_balance)	Returns a "05_{username}...{balance}" transaction line.
	log_transaction(transaction_output)	Appends the transaction line to self.transaction_file.
Delete (in delete.py)	__init__(userType, accounts, transaction_file="daily_transaction_file.txt")	Constructor: sets userType, references accounts, sets transaction file.

	process_deletion()	Admin-only. Prompts for account holder name and account number, and logs the deletion transaction.
	return_transaction_output(deleted_account)	Returns a "06_{username}_{account_number}_00000.00__" line.
	log_transaction(transaction_output)	Appends the deletion line to self.transaction_file.
ChangePlan (in changeplan.py)	__init__(userType, user, provided_account_number, new_plan=None)	Constructor: sets user, session type, optional new plan, etc.
	process_changeplan()	Admin-only. Validates the account is active, correct number, toggles or sets plan from SP↔NP.
	return_transaction_output()	Returns an "08_{username}_NNNNN_00000.00_{plan}" line.
Disable (in disable.py)	__init__(userType, provided_account_holder, provided_account_number, users)	Constructor: references the userType, name, account number, and a dict of all user accounts.
	process_disable()	Admin-only. Verifies the holder name matches account. Checks if active, then sets availability to "D".
	return_transaction_output()	Returns "07_{username}_{account_number}_00000.00_D" or empty if failed.