Assignment 3: Professional Documentation Presentation

BANK USER MANAGEMENT SYSTEM

Declan W & Earvin T

06/12/2024

Project Purpose & Features Overview

- Purpose:
  - Simulate essential banking operations for user registration, deposits, and withdrawals.
  - Provide a practical learning experience for Python developers
  - Emphazise JSON-based data storage

- Features:
  - Deposit: Allows users to deposit money into specified accounts.
  - Withdraw: Allows users to withdraw money from specified accounts.
  - Show accounts: Allows users to show information on accounts in their name.
  - Add User: Creates user instance to allow users to perform operations.

Users Class

- Purpose: This class is designed to store and manage user-related information, including personal details and the accounts associated with each user. It facilitates user operations like adding accounts and managing account activities.

- Attributes:
  - firstname: The first name of the user.
  - lastname: The last name of the user.
  - dobm, dobd, doby: The user's date of birth (month, day, year).
  - accounts: A list of accounts associated with the user.
- Methods:
  - add_account(account_type, id): Adds a new account to the user's list of accounts.
  - deposit(amount, account_index): Deposits a specific amount into one of the user’s accounts.
  - withdraw(amount, account_index): Withdraws a specific amount from one of the user’s accounts.
  - get_accounts(): Displays all of the user’s accounts with their details.
  - compound_accounts(): Applies interest rates to all of the user's accounts.

Users Class

- Methods:
  - add_account(account_type, id): Adds a new account to the user's list of accounts.
  - deposit(amount, account_index): Deposits a specific amount into one of the user’s accounts.
  - withdraw(amount, account_index): Withdraws a specific amount from one of the user’s accounts.
  - get_accounts(): Displays all of the user’s accounts with their details.
  - compound_accounts(): Applies interest rates to all of the user's accounts.

Feedback & Implementation

- **Highlighted Feedback From Ben Gorman:**
- Comments on classes, functions, and packages were incomplete and inconsistently placed.
- Ethical concerns about privacy policies and account security risks.
- Highlighted from Chris:
- Visuals on README.
- Details on Features.

- **Actions Taken:**
- Added detailed and consistent comments for all classes, functions, and packages.
- Reorganized comments to follow standard conventions.
- Addressed ethical concerns with disclaimers (covered by Earvin).

- **Impact:**
- "Improved code readability and aligned with professional standards."

Ethical Issues

- Disclaimer: At this present moment this application is not an official banking app for commercial use.
- Used for educational purposes for student developers
- Needs implementation of passwords and user authentication if it were to be an official banking app.

Accounts Class

- Purpose: This class stores information of an account such as balance, rate and perform operations on the balance.

- Attributes:
  - account_type : if it is a 'checking' or 'savings' account
  - balance: the balance of the account
  - id: Unique ID of the account
  - rate: assigned at creation depending on the account type

Accounts Class

- Methods:
  - deposit: Adds amount from balance specified by user into account balance
  - withdraw: Subtracts amount from balance specified by user from account.
  - compound (not yet functional): Multiplies rate over time to balance.

```
def deposit(self, amount):
    # Checks for valid deposit amount, must be positive
    if amount > 0:
        # Adds input to current balance
        self.balance += amount
    else:
        # Prints if not valid
        print(f"{error_color}Deposit amount must be positive.{reset}")

def withdraw(self, amount):
    # Checks for valid withdrawl amount, must be positive
    if 0 < amount <= self.balance:
        # Subtracts input amount from balance
        self.balance -= amount
        # Prints if valid withdrawal
        print(f"\n{successful_input_color}Withdrawal successful{reset}\n")
    else:
        # Prints if not valid amount
        print(f"\n{error_color}Invalid withdrawal amount.{reset}")

def compound(self):
    # Multiplies rate to account
    self.balance *= self.rate
```

Colored

- We decided to use the colored external package to help highlight printed lines on the terminal.
- Highlights prompts, successful operations, errors, outputs.
- Imported in one module, stored as strings then imported in the main file.

```python
from colored import Fore, Back, Style # type: ignore

# Different colors to call for printing on console
# Color of error message
error_color: str = f"{Back.red}{Style.underline}{Style.bold}"
# Shortcut for style reset
reset: str = f"{Style.reset}"
# Color of checking account info
checking_account_color: str = f"{Back.blue}"
# Color of saving account info
saving_account_color: str = f"{Back.yellow}"
# Color of prompt message
propmt_color: str = f"{Back.red}"
# Color of successful operation
successful_input_color: str = f"{Back.green}"
```

```python
action = input(f"{propmt_color}Choose an action:{reset}\n(1) Deposit (2) Withdraw (3) Show Accounts (4) Add User (5) Exit:\n")
```

```
Choose an action:
(1) Deposit (2) Withdraw (3) Show Accounts (4) Add User (5) Exit:
```

Conclusion

Contact & Project Information

Github Repository: https://github.com/declan-whitty/bank-user-management-system

Team Members: Declan Whitty & Earvin Tumpao