Module-4: Portfolio Milestone

Alejandro Ricciardi

Colorado State University Global

CSC372: Programming 2

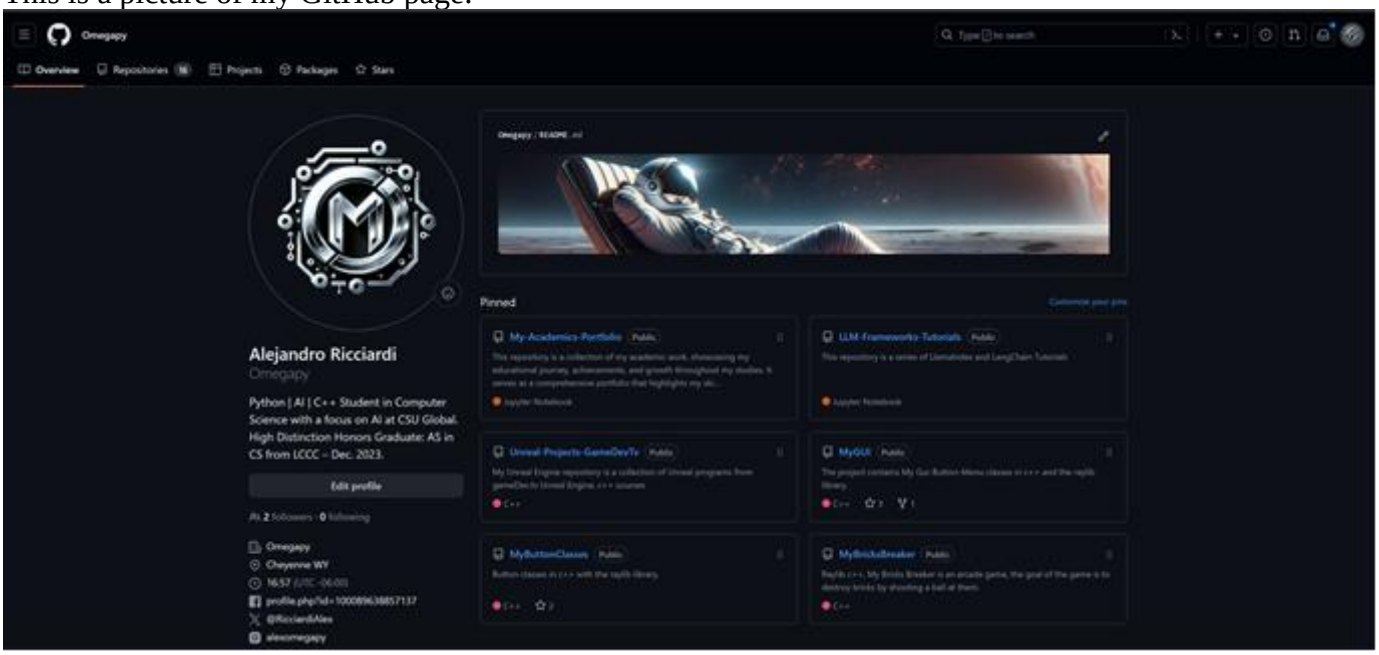
Dr. Vanessa Cooper

July 7, 2024

Module-4: Portfolio Milestone

This documentation is part of Module 4: Portfolio Milestone Assignment from CSC372: Programming 2 at Colorado State University Global. The assignment asks for corrections to assignments from Weeks 1-3. If no corrections are required, then select one assignment, discuss how the code can be utilized in a larger application, and include a UML diagram of that larger, albeit simple, application. For this assignment, I do not have corrections that are required to be made to my critical thinking assignments from weeks 1-3. Thus, I selected my assignment from week 2, Critical Thinking 2 – Bank Account GUI, to discuss how the code can be utilized in a larger application, and to include a UML diagram of that larger, albeit simple, application.

Git Repository
This is a picture of my GitHub page:



I use GitHub as my Distributed Version Control System (DVCS), the following is a link to my GitHub, Omegapy.
My GitHub repository that is used to store this assignment is named My-Academics-Portfolio and the link to this specific assignment is: https://github.com/Omegapy/My-Academics-Portfolio/tree/main/Programming-2-CSC372/Module-4%20Portfolio%20Milestone

Critical Thinking 2 – Bank Account GUI

Bank Account GUI is a simple banking manager Java program that utilizes the swing library, a graphical user interface (GUI) library. The program manages bank accounts and checking accounts with various functionalities such as creating accounts, attaching checking accounts, depositing and withdrawing funds, and viewing account balances.
Bank Account GUI Classes Description:

- The BankAccount Class:
  The BankAccount class instantiates a bank account object with fields for first name, last name, account ID, and balance. It provides methods to deposit and

withdraw funds and does not allow cash overdrafts; however, its extended CheckingAccount class allows overdrafts with an overdraft fee.

- The CheckingAccount Class:
  The CheckingAccount class instantiates a checking account object, it is an extension of the BankAccount class. It includes an interest rate and allows for overdraft withdrawals but it applies an overdraft fee.
- FrameBankAccount:
  FrameBankAccount class utilizes the swing library to create a GUI that manages bank accounts. It provides the user with various methods to manage bank accounts. Methods such as adding new bank accounts, attaching checking accounts, depositing, and withdrawing funds, and viewing account balances.
- The Main Class:
  The Main class initializes and runs the Bank Account GUI application.
  It also populates the system with some fake checking accounts and launches the GUI.

Utilization in a Larger Bank Application:

The Bank Account GUI program could be utilized in a larger banking application. Its implementation could be done as follows:

- By implementing a new Customer class that instantiates customers' objects with personal information.
- By implementing a new a CustomerManagement class to handle customer registration, updates, and queries.
- By modifying the FrameBankAccount class integrate the new Customer and CustomerManagement classes.
- By modifying the BankAccount class into an abstract Account class that serves as a parent for all the different bank account types.
- By extending the Account class with the CheckingAccount, SavingsAccount, and a new Loan classes.
- By implementing a new Transaction class to store accounts’ activities (deposits, withdrawals).
- By implementing a new Transactions class to manage and store Transaction objects.
- By implementing a new Loan class that is implemented by a LoanApplication class.
- By modifying the FrameBankAccount class to include the new classes and functionalities

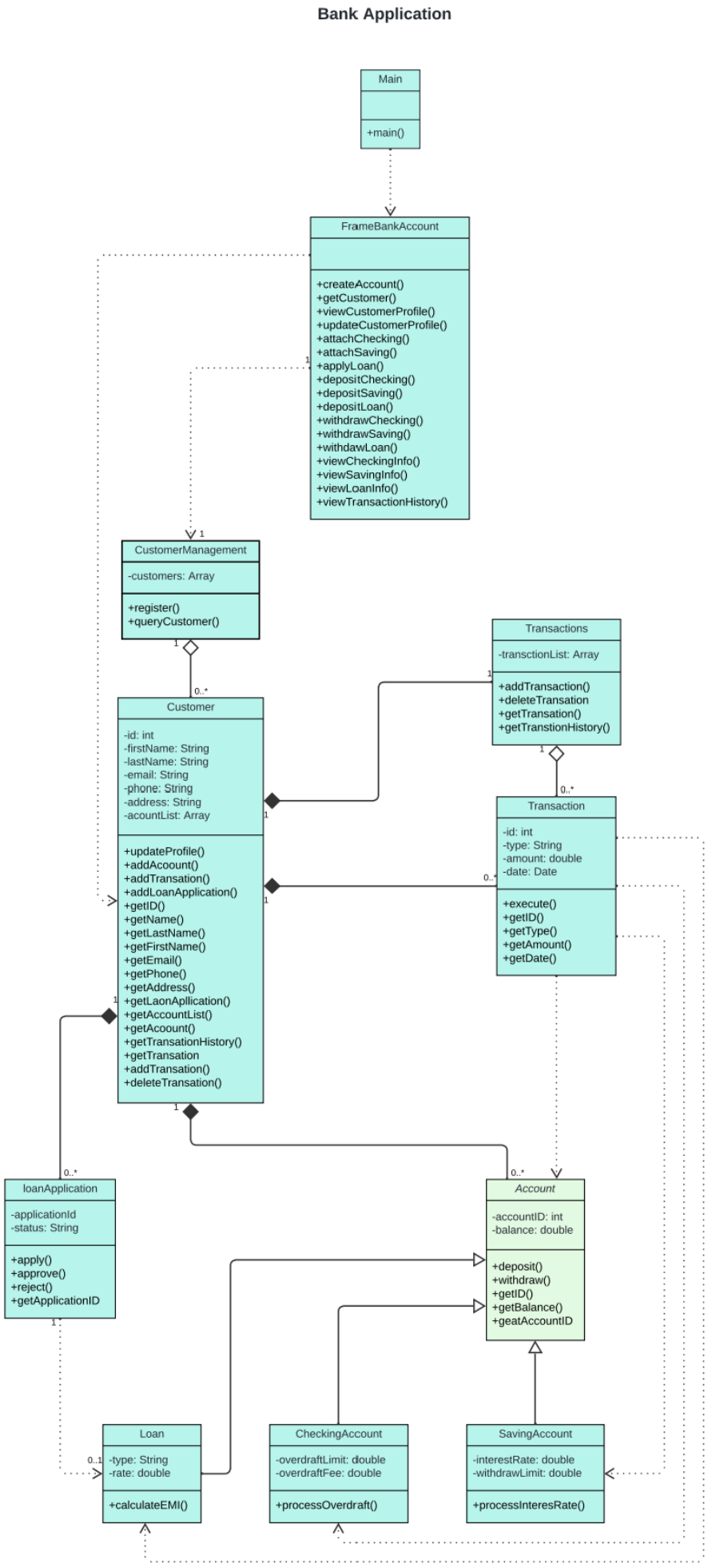
Bank Application Classes:

- **Main**
  Description: The entry point of the application.
  Relationships:
  - Uses FrameBankAccount
- **FrameBankAccount**
  Description: The main GUI class for account management.
  Relationships:
  - Uses Customer and CustomerManagement
- **CustomerManagement**
  Description: Manages customer data.
  Relationships:

    - Aggregates Customer(s)
    - Used by FrameBankAccount
- **Customer**
  Description: instantiates customer object that stores customer personal information.
  Relationships:
    - Composes Account, Transactions, Transactions, and loanApplication
    - Aggregated by CustomerManagement
- **Account** (Abstract)
  Description: Parent class for all the different types of bank accounts.
  Relationships:
    - Composed by Customer
    - Parent of CheckingAccount, SavingAccount, and Loan
- **CheckingAccount**
  Description: Instantiates checking account objects.
  Relationships:
    - Extends Account
    - Used by Transaction
- **SavingsAccount**
  Description: Instantiates saving account objects.
  Relationships:
    - Extends Account
    - Used by Transaction
- **Loan Description**:
  Description: Instantiates loan account objects.
  Relationships:
    - Extends Account
    - Used by Transaction and LoanApplication
- **Transaction**
  Description: Instantiates accounts transaction objects.
  Relationships:
    - Aggregated by Transactions
    - Uses CheckingAccount, SavingAccount, and Loan
- **Transactions**
  Description: Manages and stores a transaction objects.
  Relationships:
    - Composes by Customer
    - Aggregates Transaction
- **LoanApplication**: Instantiates applying for a loan objects.
  Relationships:
    - Composed by Customer
    - Uses Loan

The Bank Account GUI program was expanded and integrated into a larger Bank application by adding various classes and functionality, and by changing the BankAccount class to the Account class, an abstract class that serves as the base for all account types including CheckingAccount, SavingsAccount, and Loan

UML Diagram:


Bank Application
Main
+main()
FrameBankAccount
+createAccount()
+getCustomer()
+viewCustomerProfile()
+updateCustomerProfile()
+attachChecking()
+attachSaving()
+applyLoan()
+depositChecking()
+depositSaving()
+depositLoan()
+withdrawChecking()
+withdrawSaving()
+withdawLoan()
+viewCheckingInfo()
+viewSavingInfo()
+viewLoanInfo()
+viewTransactionHistory()
CustomerManagement
-customers: Array
+register()
+queryCustomer()
Transactions
-transctionList: Array
+addTransaction()
+deleteTransation
+getTransation()
+getTranstionHistory()
Customer
-id: int
-firstName: String
-lastName: String
-email: String
-phone: String
-address: String
-acountList: Array
+updateProfile()
+addAcoount()
+addTransation()
+addLoanApplication()
+getID()
+getName()
+getLastName()
+getFirstName()
+getEmail()
+getPhone()
+getAddress()
+getLaonApllication()
+getAccountList()
+getAcoount()
+getTransationHistory()
+getTransation
+addTransation()
+deleteTransation()
Transaction
-id: int
-type: String
-amount: double
-date: Date
+execute()
+getID()
+getType()
+getAmount()
+getDate()
loanApplication
-applicationId
-status: String
+apply()
+approve()
+reject()
+getApplicationID
Account
-accountID: int
-balance: double
+deposit()
+withdraw()
+getID()
+getBalance()
+geatAccountID
Loan
-type: String
-rate: double
+calculateEMI()
CheckingAccount
-overdraftLimit: double
-overdraftFee: double
+processOverdraft()
SavingAccount
-interestRate: double
-withdrawLimit: double
+processInteresRate()