CodeHub - Project Report

by Muhammad Abduh

UNIVERSITI TEKNOLOGI MALAYSIA

Object Oriented Programming (SECJ2154)

SEMESTER 2 2023/2024

GROUP PROJECT

Personal Finance Manager

AIDAN ANDREW (A22EC0036)

MUHAMMAD ABDUH BIN ABDUL BA'ARI (A22EC0199)

MOHAMMAD SYAKIRIN BIN MOHAMMAD YUSOF (A22EC0195)

MUHAMMAD HAZIQ BIN SAAMSOL (A22EC0214)

2 SECRH

SECTION 04

Lecturer:

MADAM LIZAWATI MI YUSUF

18 JUNE 2024

SECTION A: PROJECT DESCRIPTION

Synopsis:
Our project, the Finance Manager Project, aimed to assist users in efficiently managing their personal accounts. The system has features for tracking transactions, savings, bank accounts, budgeting, and report generation. Multiple accounts can be created and managed, spending can be tracked, budget limitations can be set, and savings targets can be kept an eye on. The method facilitates the division of transactions into distinct categories, such as groceries and shopping, which enables thorough financial analysis. The system's design makes it easy for users to navigate between its various functionalities, making it user-friendly.

Objective and Scope:
The primary objective of the project is to provide users with a user friendly system to manage their finances. The scope are:
- Manage multiple bank accounts simultaneously
- Track and categorize transactions
- Setting and monitoring budget
- Create savings goals
- Analyze financial data based on generated reports
- Protected users accounts data

1

Workflow:
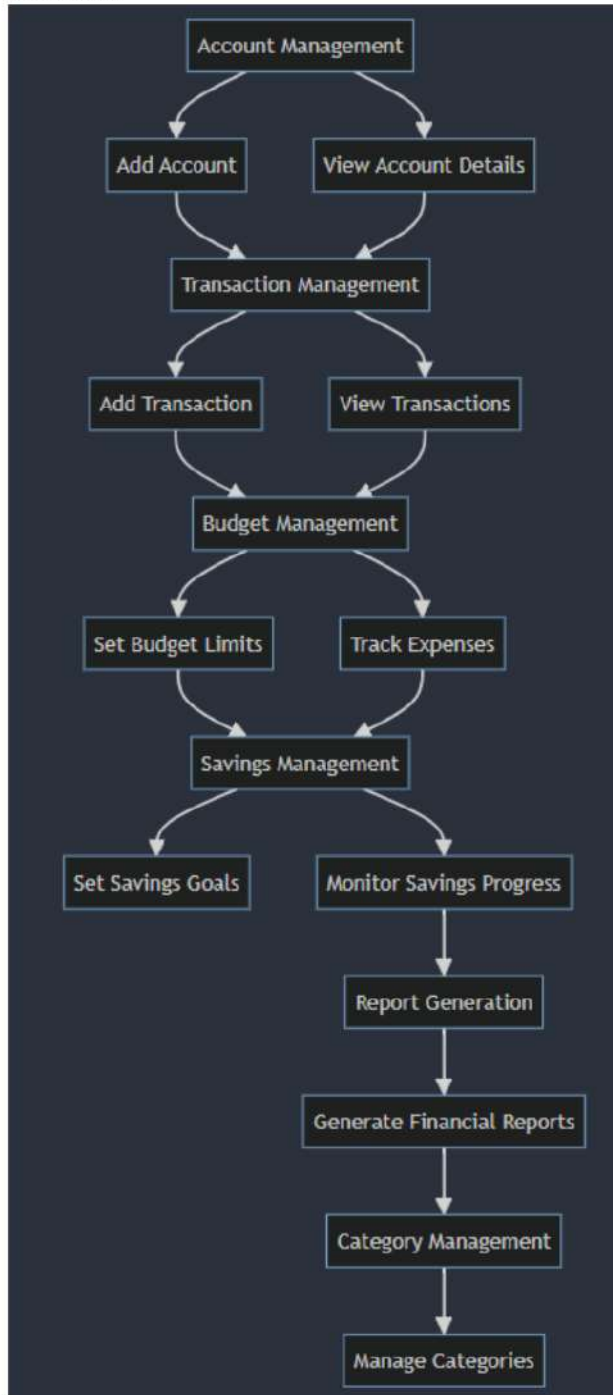
OO Concepts:

1. Encapsulation:
   - The Account class encapsulates account details like name, id and balance and provides public methods like getBalance(), deposit() and withdraw().
2. Inheritance:
   - The ShoppingCategory and FoodCategory class is derived from the Category class.
3. Association:
   - The Account class has zero-to-many relationship with the class Budget.
4. Aggregation:
   - The Bank class has an aggregation relationship with class Account, as there is an Account object inside the Bank class.
5. Composition
   - The Account class has a composition relationship with class Transaction, as there is a Transaction method inside the Account class.
6. Polymorphism
   - The ShoppingCategory, FoodCategory, Deposit and OtherCategory class override the method getCategoryType() from the Category class to behave differently in respective classes.

SECTION B: CLASS DIAGRAMS

Bank
- name : String
- account : Vector <Account>
+ Bank (name : String)
+ getName () : String
+ getAccount () : Account
+ addAccount (account : Acc...

<<interface>> Taxable
+ FOOD_TAX = 0.06 : double
+ SHOP_TAX = 0.1 : double
+ calculateTax(double) : double

Saving
- id : int
- name : String
- targetAmount : double
- currentAmount : double
- targetDate : Date
+ Saving(id:int, name:String, targetAmount : double, targetDate:Date) : int
+ getId () : int
+ getName () : String
+ getTargetAmount () : double
+ getCurrentAmount () : double
+ getTargetDate () : Date
+ addAmount(amount : double) : void

Account
- id : int
- name : String
- balance : double
+ Account (id : int, name :String, balance : double)
+ getId () : int
+ getName () : String
+ getBalance () : double
+ getBank () : Bank
+ getTransaction () : Vector<Transaction>
+ getBudgets () : Vector<Budget>
+ getSavings () : Vector<Saving>
+ addBank (bank : Bank) : void
+ deposit (balance : double) : void
+ withdraw (balance : double) : void
+ addTransaction (id : int, name : String, balance : do...
+ addbudget (limit : double, category : Category : voi...
+ removeBudget(budget : Budget) : void
+ addSaving (id: int, name : String, targetAmount : do...
+ getBudgetByCategory (category : Category) : Budg...
+ getTransactionDetails () : String
+ displayBudget() : void
+ display(info() : void
+ addBank(Bank bank) : void

Transaction
- id : int
- description : String
- amount : double
- date : Date
- category : Category
+ Transaction (id : int, description :String ,amount :double, date : Date, category : Category)
+ getId () : int
+ getDescription () : String
+ getAmount () : double
+ getDate () : Date
+ getCategory () : Category
+ setCategory (category : Category) : void
+ getDetails (rate : double) : String

<> Category
- id : int
- name : String
- description : String
+ Category (id : int, name : String, description : String)
+ getId () : int
+ getName () : String
+ getDescription () : String
+getCategoryType() : String

ShoppingCategory
+ ShoppingCategory (id :int)
+ getCategoryType() : String

FoodCategory
+ FoodCategory (id :int)
+ getCategoryType() : String

Deposit
+ Deposit (id :int)
+ getCategoryType() : String

OtherCategory
+ OtherCategory(id:int)
+ getCategoryType() : String

Users
- id : int
- name : String
+ Users(id : int , name : String (
+ getId () : int
+ getName () : String
+ getAccounts () : Vector <Account>
+ addAccount (account : Account) : int

Budget
- limit : double
- currentExpense : double
+ Budget (limit : double , category : Category)
+ getLimit () : double
+ setLimit (limit : double) : void
+ getCurrentExpense () : double
+ setCurrentExpense (currentExpense : double) :
+ getCategory () : Category
+ setCategory (category : Category) : void
+ display() : void

Report
- users : User
- exchangeRate : double
+ Report (exchangeRate : double)
+ setUser (user : User) : void
+ setExchangeRate (exchangeRate : double) : void
+ displayAccountBalanceAndTransaction () : void
+ displayAllInfo () : void

Main
+ main (args: String []) : void
- printMenu() : void
- addAccount(scanner : Scanner bank : Bank , user : Users) : void
- depositMoney(scanner : Scanner, user : Users) : void
- withdrawMoney(scanner : Scanner, user : Users) : void
- addBudget (scanner : Scanner, user :Users, categories : Vector<Category>) : void
- addSaving (scanner : Scanner, user:Users) void)
- addTransaction (scanner : Scanner, user:Users, categories :Vector <Category>) : void
- findAccountById (accounts : Vector<Account> , id:int) : Account
- findCategoryById (categories : Vector<Category> , id:int) : Category

4

Class Users:

Attributes	Description
id	Unique id for each user
name	Name of the user
account	List of accounts user have
Methods	**Description**
Users	To initialize the user instance
getId	To retrieve user id
getName	To retrieve user name
getAccount	To retrieve user list of account
addAccount	To add account into the list of account

Class Bank:

Attributes	Description
name	Name of the Bank
account	List of accounts associate to the bank
Methods	**Description**
Bank	To initialize the Bank instance
getName	To retrieve Bank name
getAccount	To retrieve user list of account
addAccount	To add account into the list of account

Class Budget:

Attributes	Description
limit	The budget limit of the category assign to
currentExpense	Total expense of that category when budget is assign
category	Category that assign to budget
Methods	**Description**
Budget	To initialize the Budget instance
getLimit	To retrieve Budget limit
setLimit	To set limit of the budget
getCurrentExpence	To retrieve the current Expense of that category in which the budget has been set.
setCurrentExpense	To set current expense of the budget
getCategory	To retrieve Categories that the budget has.
setCategory	To set Category to the budget
display	To display budget information

Class Saving:

Attributes	Description
id	An unique Id to identified the Saving instance
name	Name of the saving account
targetAmount	A goal amount for the Saving
currentAmou nt	Current amount in saving account
targetDate	A date which the saving account goal is achieve

Methods	Description
Saving	To initialize the Saving instance
getId	To retrieve Saving instance id
getName	To retrieve Saving name
getTargetAmo unt	To retrieve the target amount for the savings account.
getCurrentAm ount	To retrieve the current amount of the saving account
getTargetDate	To retrieve target date for the saving account
addAmount	To add an contribution amount to the saving account

Class Report:

Attributes	Description
users	A user instance that associate to the report instance
exchangeRate	Rate of the currency

Methods	Description
Report	To initialize the Report instance
setUser	To add associate user to the report instance
setExchangeR ate	To set an exchange rate
displayAccoun tBalanceAndT ransaction	To display each account and their transaction
displayAllInfo	To display all info of user including account,budget, saving and transaction

Class Category:

Attributes	Description
id	Identification for category
name	Category name
description	Category description
Operations	Description
Category	To initialize Category instance
getId	To retrieve id of the category
getName	To retrieve name of the category
getDescription	To retrieve description of the category
getCategoryType	To retrieve the category type

Class Taxable:

Attributes	Description
FOOD_TAX	Tax for food category
SHOP_TAX	Tax for Shopping category
OTHER_TAX	Tax for Other category
Operations	Description
calculateTax	To calculate total tax by multiplying the amount with the tax based on their category

Class Transaction:

Attributes	Description
id	Id of the Transaction
amount	Amount of the transaction
description	Description of the category
date	Date when the transaction was made
category	Category of the transaction is associate
Operations	Description
Transaction	To initialize Transaction instance
getId	To retrieve id of the transaction
getAmount	To retrieve the amount of the transaction
getDescription	To retrieve description of the transaction
getDate	To retrieve the date of the transaction
getCategory	To retrieve the category associate with the transaction
setCategory	To set category to the transaction
getDetails	To display all the details of the transaction

Class Account:

7

Attributes	Description
id	Id of the Account
name	Name of the Account
balance	The balance of the Account
bank	Bank of the Account is associate
transactions	List of transaction the Account has
budgets	List of budget the Account has
savings	List of saving the Account has
Operations	**Description**
Account	To initialize Account instance
getId	To retrieve id of the Account
getName	To retrieve the name of Account
getBalance	To retrieve balance amount of the Account
getBank	To retrieve the Bank associate to the Account
getTransaction	To retrieve list of transaction made by the Account
getBudgets	To retrieve list of budgets that Account have
getSaving	To retrieve list of Saving that Account have
setBank	To set bank to associate to the Account
addBank	To add bank to associate with the Account
deposit	To add money into the bank
withdraw	To take out money from the Account
addTransaction	To add transaction into the list of transaction
addBudget	To add Budget into the list of budget
removeBudget	To remove budget from the Account
addSaving	To add Saving into the list of saving
getBudgetByCategory	To retrieve category which the budget is associate
getTransactionDetails	To display transactions details
displayBudget	To display budgets details
displayInfo	To display Account information

Class Deposit

Attributes	Description
id	Specific identification for deposit
name	Deposit category
description	Description of the category with value "Deposit"
Operations	**Description**
Category	To initialize Category instance
getId	To retrieve id of the category
getName	To retrieve name of the category
getDescription	To retrieve description of the category
getCategoryType	To retrieve the category type which is "Deposit"
calculateTax	To calculate tax regarding deposit

Class Main:

Attributes	Description
report	To store user information
categories	To initialize and store the categories
scanner	To read input from user
inp	To read bank file
acc	To read account file
bud	To read budget file
trans	To read transaction file
sav	To read saving file
bank1	To initialize the bank
user1	To initialize the user
Operations	**Description**
Main	The program start
printMenu	To display the main menu
addAccount	Let user to insert and create an Account
depositMoney	Let User to key in money into the Account
withdrawMoney	Let user to take out money from the Account
changeCurrency	Let user to change currency rate
addBudget	Let user to add budget to category
addSaving	Let user to add Saving account
addTransaction	Let user to add transaction into account to track
findAccountById	To find account by key of ID
findCategoryById	To find category by key of ID
writeBankFile	To write bank details to file
writeAccountFile	To write account details to file
writeBudgetFile	To write budget details to file
writeTransactionFile	To write transaction details to file
writeSavingFile	To write saving details to file

Class ShoppingCategory:

Attributes	Description
id	Id for shopping
name	Category name shopping
description	Description regarding shopping
Operations	Description
Category	To initialize Category instance
getId	To retrieve id of the category
getName	To retrieve name of the category
getDescription	To retrieve description of the category
getCategoryType	To retrieve the category type which is "Shopping"
CalculateTax	To calculate tax regarding shopping tax

Class FoodCategory:

Attributes	Description
id	Unique identification for food
name	Food category
description	Food description
Operations	Description
Category	To initialize Category instance
getId	To retrieve id of the category
getName	To retrieve name of the category
getDescription	To retrieve description of the category
getCateogryType	To retrieve the category type which is "Food"
calculateTax	To calculate tax regarding food tax

Class PauseScreen:

Attributes	Description
Operations	Description
ClearScreen	To clear the screen
pauseScreen	To make loading screen

Class AccountNotFoundException:

Attributes	Description
message	To describe the exception
Operations	Description
AccountNotFoundException	To initialize the instance

Class InsufficientFundsException:

Attributes	Description
message	To describe the exception
Operations	**Description**
InsufficientFundsException	To initialize the instance

Class CategoryNotFoundException:

Attributes	Description
message	To describe the exception
Operations	**Description**
CategoryNotFoundException	To initialize the object for this particular class

SECTION C: SOURCE CODE AND USER MANUAL

class Account.java

```java
import java.util.*;
import java.sql.Date;



class Account {
    private int id;
    private String name;
    private double balance;
    private double rate;
    private Bank bank;
    private Vector<Transaction> transactions;
    private Vector<Budget> budgets;
    private Vector<Saving> savings;
    Scanner scanner = new Scanner(System.in);


    public Account(int id, String name, double balance) {
        this.id = id;
        this.name = name;
        this.balance = balance;
        this.transactions = new Vector<>();
        this.budgets = new Vector<>();
        this.savings = new Vector<>();
        rate = 1;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getBalance() {
        return balance;
```

12

```java
    }

    public Bank getBank() {
        return bank;
    }

    public Vector<Transaction> getTransactions() {
        return transactions;
    }

    public Vector<Budget> getBudgets() {
        return budgets;
    }

    public Vector<Saving> getSavings() {
        return savings;
    }

    // association to bank class
    public void setBank(Bank bank) {
        this.bank = bank;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) throws
InsufficientFundsException {
        if (balance < amount) {
            throw new InsufficientFundsException("Insufficient
funds.");
        }
        balance -= amount;
    }

    public void addTransaction(int id, String name, double
balance, Date date, Category category) {
        Transaction obj = new Transaction(id, name, balance, date,
category);
        transactions.add(obj);
        obj.setCategory(category);
```

13

```java
    }

    public void addBudget(double limit, Category category) {
        budgets.add(new Budget(limit, category));
    }

    public void removeBudget(Budget budget) {
        budgets.remove(budget);
    }

    public void addSaving(Saving s) {
        savings.add(s);
    }

    public Budget getBudgetByCategory(Category category) {
        for (Budget budget : budgets) {
            if (budget.getCategory().getId() == category.getId())
{

                return budget;
            }
        }
        return null;
    }

    public String getTransactionDetails() {
        double bal = balance * rate;
        StringBuilder details = new StringBuilder(
                "Account ID: " + id + ", Name: " + name + ", 
Balance: RM" + bal + "\n");
        for (Transaction transaction : transactions) {

details.append(transaction.getDetails(rate)).append("\n");
        }
        return details.toString();
    }

    public void displayBudget() {
        for (Budget budget : budgets) {
            budget.display();
            System.out.println();
        }
    }
```

```java
    public void displayInfo() {
        System.out.printf("%-15s%-15s%-5s\n" , "Account ID",
"Name", "Balance");
        System.out.printf("%-15s%-15s%-5.2f\n", id,name,balance *
rate);
    }
    public void addBank(Bank bank){
        this.bank = bank;
    }
}
```

Class Bank.java

```java
import java.util.*;

public class Bank {
    private String name;
    private Vector<Account> accounts = new Vector<>();

    public Bank(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void addAccount(Account account) {
        accounts.add(account);
    }

    public Account getAccount(int id) throws
AccountNotFoundException {
        for (Account account : accounts) {
            if (account.getId() == id) {
                return account;
            }
        }
        throw new AccountNotFoundException("Account with ID " + id
+ " not found.");
    }
}
```

16

Class Budjet.java

```java
public class Budget {
    private double limit;
    private double currentExpense;
    private Category category;

    public Budget(double limit, Category category) {
        this.limit = limit;
        this.category = category;
        this.currentExpense = 0.0;
    }

    public double getLimit() {
        return limit;
    }

    public void setLimit(double limit) {
        this.limit = limit;
    }

    public double getCurrentExpense() {
        return currentExpense;
    }

    public void setCurrentExpense(double currentExpense) {
        this.currentExpense = currentExpense;
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    public void display() {
        System.out.println("Budget for " +
category.getCategoryType() + ":>");
        System.out.println("Budget used : " + currentExpense + "/
" + limit);
```

17

```
        }
}
```

Class Category.java

```java
interface Taxable {
    double FOOD_TAX = 0.05;
    double SHOP_TAX = 0.1;
    double OTHER_TAX = 0.08;
    double calculateTax(double amount);
}

abstract class Category implements Taxable {
    private int id;
    private String name, description;

    public Category(int id, String name, String description) {
        this.id = id;
        this.name = name;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }

    public abstract String getCategoryType();
}

class ShoppingCategory extends Category {
    public ShoppingCategory(int id) {
        super(id, "Shopping", "Shopping Transaction");
    }

    public String getCategoryType() {
        return "Shopping";
    }
```

```java
        public double calculateTax(double amount) {
            return amount * SHOP_TAX;
        }
    }

class FoodCategory extends Category {
    public FoodCategory(int id) {
        super(id, "Food", "Food Transaction");
    }

    public String getCategoryType() {
        return "Food";
    }

    public double calculateTax(double amount) {
        return amount * FOOD_TAX;
    }
}

class Deposit extends Category {
    public Deposit(int id) {
        super(id, "Deposit", "Deposit Money to Account");
    }

    public String getCategoryType() {
        return "Deposit";
    }

    public double calculateTax(double amount) {
        return 0;
    }
}

class OtherCategory extends Category {
    public OtherCategory(int id) {
        super(id, "Other", "Other Transaction");
    }

    public String getCategoryType() {
        return "Other";
    }
```

```java
    public double calculateTax(double amount) {
        return amount * OTHER_TAX;
    }
}
```

class PauseScreen.java

```java
import java.util.Scanner;

class PauseScreen {

    public void ClearScreen(){
        System.out.print("\033[H\033[2J");
        System.out.flush();
    }
    public void pauseScreen(Scanner scanner) {

        System.out.println("Press Enter to continue...");
        scanner.nextLine();  // Waits for the user to press Enter

    }

}
```

Class Report.java

```java
public class Report {
    private Users user;
    private double exchangeRate; // change currency rate

    public Report(Users user ,double exchangeRate) {
        this.user = user;
        this.exchangeRate = exchangeRate;
    }

    public void setUser(Users user) {
        this.user = user;
    }

    public void setExchangeRate(double exchangeRate) {
        this.exchangeRate = exchangeRate;
        for (Account account : user.getAccounts()) {
            account.setRate(exchangeRate);
        }
    }

    public void displayAccountBalancesAndTransactions() {
        for (Account account : user.getAccounts()) {
            account.displayBudget();
            System.out.println(account.getTransactionDetails());
```

22

```java
        }
    }

    public void displayAllInfo() {
        System.out.println("\nUser Information:");
        System.out.println("ID: " + user.getId());
        System.out.println("Name: " + user.getName());
        System.out.println();

        for (Account account : user.getAccounts()) {
            System.out.println("Account Information:");
            System.out.println("ID: " + account.getId());
            System.out.println("Name: " + account.getName());
            System.out.println("Balance: RM" +
account.getBalance() * exchangeRate);
            System.out.println();

            System.out.println("Budget Information:");
            account.displayBudget();
            System.out.println();

            System.out.println("Saving Information:");
            for (Saving saving : account.getSavings()) {
                System.out.println("ID: " + (saving.getId()+1));
                System.out.println("Name: " + saving.getName());
                System.out.println("Target Amount: RM" +
saving.getTargetAmount());
                System.out.println("Current Amount: RM" +
saving.getCurrentAmount());
                System.out.println("Target Date: " +
saving.getTargetDate());
                System.out.println();
            }

            System.out.println("Transaction Information:");
            System.out.println(account.getTransactionDetails());
            System.out.println();
        }
    }
}
```

Class Saving.java

```java
import java.sql.Date;

public class Saving {
    private int id;
    private String name;
    private double targetAmount, currentAmount;
    private Date targetDate;

    public Saving(int id, String name, double targetAmount, double
currentAmount, Date targetDate) {
        this.id = id;
        this.name = name;
        this.targetAmount = targetAmount;
        this.currentAmount = currentAmount;
        this.targetDate = targetDate;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getTargetAmount() {
        return targetAmount;
    }

    public double getCurrentAmount() {
        return currentAmount;
    }

    public Date getTargetDate() {
        return targetDate;
    }

    public void addAmount(double amount) {
        currentAmount += amount;
    }
```

```
}
```

Class Transaction.java

```java
import java.sql.Date;
import java.util.*;


class Transaction {
    private int id;
    private String description;
    private double amount;
    private Date date;
    private Category category;
    Scanner scan = new Scanner(System.in);

    public Transaction(int id, String description, double amount,
Date date, Category category) {
        this.id = id;
        this.description = description;
        this.amount = amount;
        this.date = date;
        this.category = category;
    }

    public int getId() {
        return id;
    }

    public String getDescription() {
        return description;
    }

    public double getAmount() {
        return amount;
    }

    public Date getDate() {
        return date;
    }
```

```java
    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    public String getDetails(double rate) {
        return "Transaction ID: " + id + ", Description: " +
description + ", Amount: RM" + (amount * rate) + ", Date: " + date
            + ", Category: " + category.getName();
    }

    public int getCategoryId(){
        if(category.getName().equals("Shopping")){
            return 1;
        }
        else if(category.getName().equals("Food")){
        return 2;
        }
        else{
            return 3;
        }
    }
}
```

Class Users.java

```java
import java.util.Vector;

public class Users {
    private int id;
    private String name;
    private Vector<Account> accounts = new Vector<>();

    public Users(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void addAccount(Account account) {
        accounts.add(account);
    }

    public Vector<Account> getAccounts() {
        return accounts;
    }
}
```

Main Screen:
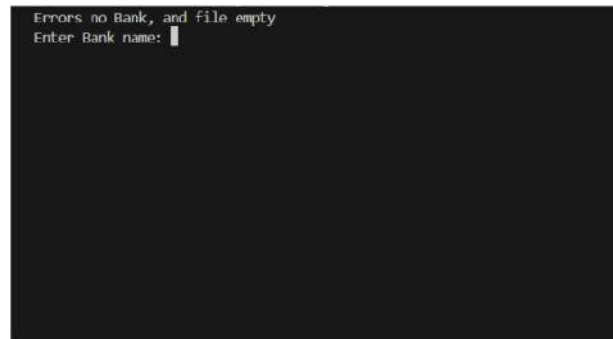
Figure 1

Figure 2

At the start there are 2 scenarios,first where you first time use the Finance Manager and the system would check if there are already registered bank,if there are not figure 1 is what you would see,you will be prompted to enter your Bank name,your ID and your Name.Figure 2 what you would see if you already use the project before.

Menu Screen 1:

Figure 3

Here you can see the main menu of the app where all the functions that you can use are as shown on Figure 3

Screen Add Account:

```
Enter account name: Amir
Enter initial balance: 1000
Account added successfully.
Account ID      Name            Balance
6               Amir            1000.00
Press Enter to continue...
```

Figure 4

After you press 1 on the main menu,you will be prompted to Enter your new Account Name,your initial Balance,after successfully created the account,the program will notify you and output your new Account along with the ID. as in Figure 4 above,press enter and you will return to main screen.

Screen Deposit Money:

```
Enter account ID: 6
Enter deposit amount: 1000
Money deposited successfully.
Press Enter to continue...
```

Figure 5

Press 2 on main Screen to deposit monet,you will be prompted to enter which account that you want to deposit the money,the amount,then you will be notified successful,press enter and you will return to main screen again.

Screen Withdraw Money:


```
Enter account ID: 6
Enter withdrawal amount: 50
PRESS 1 FOR SHOPPING
PRESS 2 FOR FOOD
PRESS 3 FOR OTHER
1
Enter Shopping name:KASUT
Money withdrawn successfully.
Press Enter to continue...
```

Figure 6

Press 3 on main screen,You again will be prompted to enter the id of account that you want to withdraw from,and the the program will ask the purpose of withdrawing money by giving you option,then the name of anything that you want to withdraw for,then you will be notified,press enter and you will return to main screen.As shown on figure 6

Screen Exchange Currency:


```
Enter account ID: 6
Enter account ID: 6
Enter new exchange rate (1 USD to target currency): 1.2
Exchange rate updated successfully.
Press Enter to continue...
```

Figure 7

Pressing 4 on main screen will allow you to change the currency of the money of youre using,it will prompt you to enter the exchange rate and it will change,noted that the program will still use RM as the currency but the value would match the exchanged rate,press enter to return to main screen.As shown on figure 7

31

Screen Display Balances and Transctions:

Figure 8

Figure 9

Figure 8 and 9 show you when you press 5 on the main Menu,it will show all the account name and balance,it will also show the each transaction that happen on each account.press enter to return to main menu.

Screen Add Budjet:


```
Enter account ID: 6
Enter budget limit: 100
Choose category: 1. Shopping
2. Food
1
Budget added successfully.
Press Enter to continue...
```

Figure 10

Pressing 6 will allow you to add a budget to your account,it will ask which account that you want to put a budget for,the budget limit and the category of the budget either Shopping or Food,then it will notify if it successful,it will also ask you to replace the budget if you already have a budget for the category,press enter to return to main menu,As shown on Figure 10

Screen Add Saving:


```
Enter account ID: 6
Enter saving goal name: RUMAH
Enter target amount: 900000
Enter current amount: 1000
Enter target date (YYYY-MM-DD): 2028-09-03
Saving goal added successfully.
Press Enter to continue...
```

Figure 11

On figure 11,after pressing 7 on the menu screen,it allow you to add saving goal,it will prompt you to enter which account you want to have a saving goal for,the name of the goal,target amount,current amount that you have and the targeted date that you want to accomplish the goal,then it will notify you if the saving are saved successfully,press enter to return to main menu.

Screen Add Transaction:


```
Enter account ID: 6
Enter transaction description: Yuran
Enter transaction amount: 250
Choose category: 1. Shopping
2. Food
1
This transaction exceeds the budget limit for the category Shopping.
Do you want to proceed? (yes/no): y
Transaction cancelled.
Press Enter to continue...
```

Figure 12

Pressing 8 will allow you to add transaction withdrawal of money,the purpose of this are for unexpected purchase,it will prompt you to enter which account to use,the amount of money,the category.It will also would check if it is out of budget,if it is it will cancel the transaction as shown on Figure 12,press enter to return to main menu.

Screen Display All:

Figure 13

35

Figure 14


```
Account Information:
ID: 4
Name: AIDAN
Balance: RM10680.0

Budget Information:

Saving Information:
Transaction Information:
Account ID: 4, Name: AIDAN, Balance: RM10680.0


Account Information:
ID: 5
Name: KKK
Balance: RM27.96

Budget Information:

Saving Information:
Transaction Information:
Account ID: 5, Name: KKK, Balance: RM27.96


Account Information:
ID: 6
Name: Amir
Balance: RM3432.0

Budget Information:
Budget for Shopping:>
Budget used : 0.0/ 100.0


Saving Information:
ID: 8
Name: RUMAH
Target Amount: RM900000.0
Current Amount: RM1000.0
```
Java: Ready

Figure 15

Figure 16

Pressing display all information on menu screen will display all the information that you have on your bank,all transactions on each account,each budget,each saving and each,it show you every action that you have done on your account each individually,press enter to return to main screen

Screen Exit:

Figure 17

Pressing 10 will exit the program,it will save all the action that you have done on the file which the program will read when you boot it up again.

37

CodeHub - Project Report

17% SIMILARITY INDEX   **13**% INTERNET SOURCES   **9**% PUBLICATIONS   **15**% STUDENT PAPERS

PRIMARY SOURCES

1	www.coursehero.com Internet Source	2%
2	Submitted to Universiti Teknologi Malaysia Student Paper	2%
3	Submitted to Study Group Australia Student Paper	1%
4	books.lihui.org Internet Source	1%
5	Submitted to University of Sydney Student Paper	1%
6	www.javanb.com Internet Source	1%
7	Submitted to Southwestern Oklahoma State University Student Paper	1%
8	Submitted to Informatics Education Limited Student Paper	1%
9	Submitted to University of Moratuwa Student Paper	1%

21 ebin.pub
    Internet Source
    <1%

22 "Pro Hibernate 3", Springer Science and
    Business Media LLC, 2005
    Publication
    <1%

23 Submitted to Astana IT University
    Student Paper
    <1%

24 Submitted to South Bank University
    Student Paper
    <1%

25 Submitted to University of Wales, Lampeter
    Student Paper
    <1%

26 library.samdu.uz
    Internet Source
    <1%

27 www2.mdpi.com
    Internet Source
    <1%

Exclude quotes   Off   Exclude matches   Off
Exclude bibliography   Off