**COMSATS University Islamabad**

**Attock Campus**Department of Electrical and Computer Engineering

**Program:** BCE-4 Spring 2021

**Lab Sessional II  
Course:** Object Oriented Programming

**Time Allowed: 1h 30min Marks: 30**

**Name: Muhammad Kaleem Ullah Reg. No: FA19-BCE-007**

**Instructor’s E-mail id**: [laraib@cuiatk.edu.pk](mailto:laraib@cuiatk.edu.pk)

**Instructions:**

1. You have to submit your answer document & code file via both MS Teams & email.
2. Save document file with your Name and Registration number.
3. No late answers would be accepted.
4. Each section of this question has separate marks so try to attempt all sections.
5. If your answer found to be plagiarized, you will get a straight off zero.
6. If your answer found to be copied from another student then marks will be divided between both.

**Question 1: [CLO-2,3,4]**

**Section 01:**

Design a generic class to hold the following information about a bank account:

* Balance of the account
* Number of deposits this month
* Number of withdrawals this month
* Annual interest rate for the account
* Monthly service charges for the account

The class should have the following member functions:

* **Constructor:** Accepts arguments for the balance and annual interest rate.
* **Deposit:** A function that accepts an argument for the amount of the deposit. The function should add the argument to the account balance. It should also increment the variable holding the number of deposits.
* **Withdraw:** A function that accepts an argument for the amount of the withdrawal. The function should subtract the argument from the balance. It should also increment the variable holding the number of withdrawals.
* **CalcInt:** A function that updates the balance by calculating the monthly interest earned by the account, and adding this interest to the balance. This is performed by the following formulas:

Monthly Interest Rate = (Annual Interest Rate / 12)

Monthly Interest = Balance \* Monthly Interest Rate

Balance = Balance + Monthly Interest

* **MonthlyProc:** A function that subtracts the monthly service charges from the balance, calls the calcInt function, and then sets the variables that hold the number of withdrawals, number of deposits, and monthly service charges to zero.

**Section 02:**

Next, design a **savings account class**, derived from the generic account class. The savings account class should have the following additional member:

* status (to represent an active or inactive account)

If the balance of a savings account falls below $45, it becomes inactive. (The status member could be a flag variable.) No more withdrawals may be made until the balance is raised above $45, at which time the account becomes active again. The savings account class should have the following member functions:

* **Withdraw:** A function that checks to see if the account is inactive before a withdrawal is made. (No withdrawal will be allowed if the account is not active.) A withdrawal is then made by calling the base class version of the function.
* **Deposit:** A function that checks to see if the account is inactive before a deposit is made. If the account is inactive and the deposit brings the balance above $45, the account becomes active again. The deposit is then made by calling the base class version of the function.
* **MonthlyProc:** Before the base class function is called, this function checks the number of withdrawals. If the number of withdrawals for the month is more than 3, a service charge of $2 for each withdrawal above 3 is added to the base class variable that holds the monthly service charges. (Don’t forget to check the account balance after the service charge is taken. If the balance falls below $45, the account becomes inactive.)

**Section 03:**

Next, design a **checking account class**, also derived from the generic account class. It should have the following member functions:

* **Withdraw:** Before the base class function is called, this function will determine if a withdrawal (a check written) will cause the balance to go below $0. If the balance goes below $0, a service charge of $12 will be taken from the account. (The withdrawal will not be made.) If there isn’t enough in the account to pay the service charge, the balance will become negative and the customer will owe the negative amount to the bank.
* **MonthlyProc:** Before the base class function is called, this function adds the monthly fee of $3, plus $0.25 per withdrawal (check written) to the base class variable that holds the monthly service charges.

**Section 04:**

Write a complete program that demonstrates these classes by asking the user to enter the amounts of deposits and withdrawals for a savings account and checking account. The program should display statistics for the month, including beginning balance, total amount of deposits, total amount of withdrawals, service charges, and ending balance.

**Note:** You can create more variables and functions other than listed above if needed.

**Solution:**

* **Code:**

#include<iostream>

using namespace std;

class Account

{

protected:

float balance,deposits,withdrawals,interestRate,serviceCharges;

public:

Account(float b=0,float i=0)

{

balance=b;

deposits=0;

withdrawals=0;

interestRate=i;

serviceCharges=0;

}

void Deposit(float d)

{

balance=balance+d;

deposits++;

}

void withdraw(float w)

{

balance=balance-w;

deposits++;

}

void calcInt()

{

float monthlyInterestRate=interestRate/12;

float monthlyInterest=balance\*monthlyInterestRate;

balance=balance+monthlyInterest;

}

void monthlyProc()

{

calcInt();

balance=balance-serviceCharges;

}

};

class Savings:public Account

{

private:

bool status;

public:

Savings()

{

if(balance<45)

{

status=false;

}

}

void withdraw(float w)

{

if (status==false)

{

cout<<"\nYour Account now is Inactive !!!";

return;

}

Account::withdraw(w);

}

void deposit(float d)

{

if(balance+d>45)

{

status=true;

Account::Deposit(d);

}

}

void monthlyProc()

{

if(withdrawals>3)

{

serviceCharges=2+serviceCharges;

if (status==false)

{

cout<<"\nYour Account now is Inactive !!!";

return;

}

}

else

{

Account::monthlyProc();

}

}

};

class Checking:public Account

{

public:

void withdraw(float w)

{

if(w<0)

{

serviceCharges=serviceCharges-12;

return;

}

else if(balance-w<0)

{

cout<<"\nThe Balance is Negative ! ";

cout<<"\nYou Own a Negative Amount To the Bank";

status=false;

}

else

{

Account::withdraw(float w);

}

}

void monthlyProc()

{

serviceCharges+=(5+(withdrawals\*0.25));

Account::monthlyProc();

}

};

int main()

{

Savings s;

Checking c;

int choice;

float amount;

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BANK ACCOUNT MENU \*\*\*\*\*\*\*\*\*\*\n";

cout<<"\n1. Saving Account Deposit. ";

cout<<"\n2. Checking Account Deposit. ";

cout<<"\n3. Savigs Account Withdrawal. ";

cout<<"\n4. Checking Account Withdrawal ";

cout<<"\n5. Exit ";

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout<<"Enter Your Choice : ";

cin>>choice;

switch (choice)

{

case 1:

cout<<"Enter Amount To Deposit : ";

cin>>amount;

s.deposit(amount);

break;

case 2:

cout<<"\nEnter Amount to deposit : ";

cin>>amount;

c.deposit(amount);

break;

case 3:

cout<<"\nEnter the Amount to withdraw : ";

cin>>amount;

s.withdraw(amount);

case 4:

cout<<"\nEnter Amount to withdraw : ";

cin>>amount;

c.withdraw(amount);

break;

default:

cout<<"\nPlease Enter the Right Option !!";

break;

}

}

* **Output:**