Lab 05

Object-Oriented Programming-2

Write a simple banking system (using OOP concepts) that allows dealing with different types of bank accounts, any account (regardless of its type) should have a customer name, id and balance.

There are two main types of accounts: saving account and checking account

Saving account: adds interest to the balance based on a pre-defined interest rate

Checking account: allows a number of free transactions (deposit and/or withdraw). After that number is exceeded, a fee is applied to each subsequent transaction

Test your system

Account.cs

```
   class Account
    {
        protected string name;
        protected int id;
        protected double balance;

        public Account(string n ,int id, double b)
        {
            this.name = n;
            this.id = id;
            this.balance = b;
        }

        public Account(string n, int id)
        {
            this.name = n;
            this.id = id;
            this.balance = 0;
        }

        public virtual bool deposit(double amount)
        {
            if (amount > 0)
            {
                this.balance += amount;
                Console.WriteLine("Successfully added {0} to
balance", amount);
                return true;
            }
            else
            {
                Console.WriteLine("Failure, invalid amount");
                return false;
            }
        }


        public virtual bool withdraw(double amount)
```

```
        {
            if (this.balance > amount)
            {
                balance -= amount;
                Console.WriteLine("Successfully withdrawn {0} off
balance", amount);
                return true;
            }
            else
            {
                Console.WriteLine("Failure, insufficient funds");
                return false;
            }
        }

        public double getBalanace()
        {
            return this.balance;
        }

        public override string ToString()
        {
            return "Account holder name: " + this.name + " - Account
ID: " + this.id + " - Available Balance: " + this.balance;
        }
    }
```

SavingAccount.cs

```
   class SavingAccount : Account
    {
        protected double interestRate;

        public SavingAccount(string n, int id, double b, double iR) :
base(n, id, b)
        {
            this.interestRate = iR;
        }

        public SavingAccount(string n, int id, double b): base(n, id,
b)
        {
            this.interestRate = 0.12;
        }

        public SavingAccount(string n, int id) : base(n, id)
        {
            this.interestRate = 0.12;
        }

        public double addInterest()
        {
            double interest = this.balance * this.interestRate;
            if (deposit(interest))
            {
                Console.WriteLine("Interest amount {0} added to
balance, new balance is : {1}", interest, getBalanace());
            }
            else
            {
                Console.WriteLine("No interest added");
            }
            return getBalanace();
        }
    }
```

CheckingAccount.cs

```
    class CheckingAccount : Account
    {
        protected static int noFreeTransactions = 3;
        protected static double extraFee = 2.5;
        protected int noTransactions;

        public CheckingAccount(string n, int id, double b):base (n,
id, b) { }
        public CheckingAccount(string n, int id): base(n, id) { }

        public override bool deposit(double amount)
        {
            bool shouldDeduceExtraFee = noTransactions + 1 >
noFreeTransactions;
            double addedBalance = shouldDeduceExtraFee ? amount -
extraFee : amount;
            if (addedBalance > 0)
            {
                this.balance += addedBalance;
                noTransactions++;
                Console.WriteLine("Successfully added " + amount + "
to balance " + ((shouldDeduceExtraFee) ? "with extra fee of " +
extraFee : " "));
                return true;
            }
            else
            {
                Console.WriteLine("Failure, invalid amount or
insufficient funds to deduce the extra fee");
                return false;
            }
        }
```

```
        public override bool withdraw(double amount)
        {

            bool shouldDeduceExtraFee = noTransactions + 1 >
noFreeTransactions;
            double deducedBalance = shouldDeduceExtraFee ? (amount +
extraFee) : amount ;

            if (this.balance > deducedBalance)
            {
                balance -= deducedBalance;
                noTransactions++;
                Console.WriteLine("Successfully deduced an amount of
" + amount + ((shouldDeduceExtraFee) ? " with an extra fee of " +
extraFee : " "));
                return true;
            }
            else
            {
                Console.WriteLine("Failure, insufficient funds");
                return false;
            }

        }
    }
```

Tester Class (Program.cs)

```
   class Program
    {
        static void Main(string[] args)
        {
            Account a1 = new CheckingAccount("Ahmed", 110, 5000);
            Console.WriteLine(a1.ToString());
            a1.deposit(20);
            a1.withdraw(10);
            a1.deposit(5);
            a1.withdraw(10);
            Console.WriteLine("Current Balance is {0}",
a1.getBalanace());


Console.WriteLine("====================================");

            SavingAccount a2 = new SavingAccount("Ali", 111, 2000,
0.1);
            Console.WriteLine(a2.ToString());
            a2.deposit(1000);
            a2.withdraw(10);
            a2.addInterest();
        }
    }
```