.NET PROGRAMMING LAB 5

Name: CH Bala Gowtham
ID: 2000032067
Section: S-13

IN-LAB:

TASK1: To create classes **Deposit** (bank account), **BaseDeposit** (regular deposit), **SpecialDeposit** (special deposit), **LongDeposit** (long-term deposit), **Client** (bank client) with set functionality.

1. To create abstract class **Deposit** and declare within it:

- Public money property only for reading **Amount** (deposit amount)
- Public integer property only for reading **Period** (time of deposit in months)
- Constructor (for calling in class-inheritor) with parameters **depositAmount** and **depositPeriod**, which creates object deposit with specified sum for specified period.
- Abstract method **Income**, which returns money value – amount of income from deposit. Income is the difference between sum, withdrawn from deposit upon expiration date and deposited sum.

2. To create classes that are inheritors of the class **Deposit**, which determine different options of deposit interest addition – class **BaseDeposit**, class **SpecialDeposit** and class **LongDeposit**. To implement in each class a constructor with parameters **amount** and **period**, which calls constructor of parent class.
3. For each inheritor class – to implement own interest addition scheme and accordingly profit margin definitions, overriding abstract method **Income** in each class.

BaseDeposit implies each month 5% of interest from current deposit sum. Each following month of income is calculated from the sum, which was received by adding to current income sum of the previous month and is rounded to hundredth.

Example: Base amount – 1000,00

In a month – 105,00; income amount – 50,00

In two months – 1102,50; income amount – 102,50

In three months – 1157,62; income amount – 157,62

SpecialDeposit implies income addition each month, amount of which (in percent) equals to deposit expiration period. If during the first month 1% is added, during the second month – 2% from the sum obtained after first month and so on.

Example: Base amount – 1000,00

In a month – 1010,00; income amount – 10,00

In two months – 1030,20; income amount – 30,20

LongDeposit implies that during first 6 months, no percent is added to client's deposit, but starting from 7th month, each month percent addition is 15% from current deposit sum, thus encouraging client to make long-term deposits.

4. To create class **Client** (bank client) and declare within it:

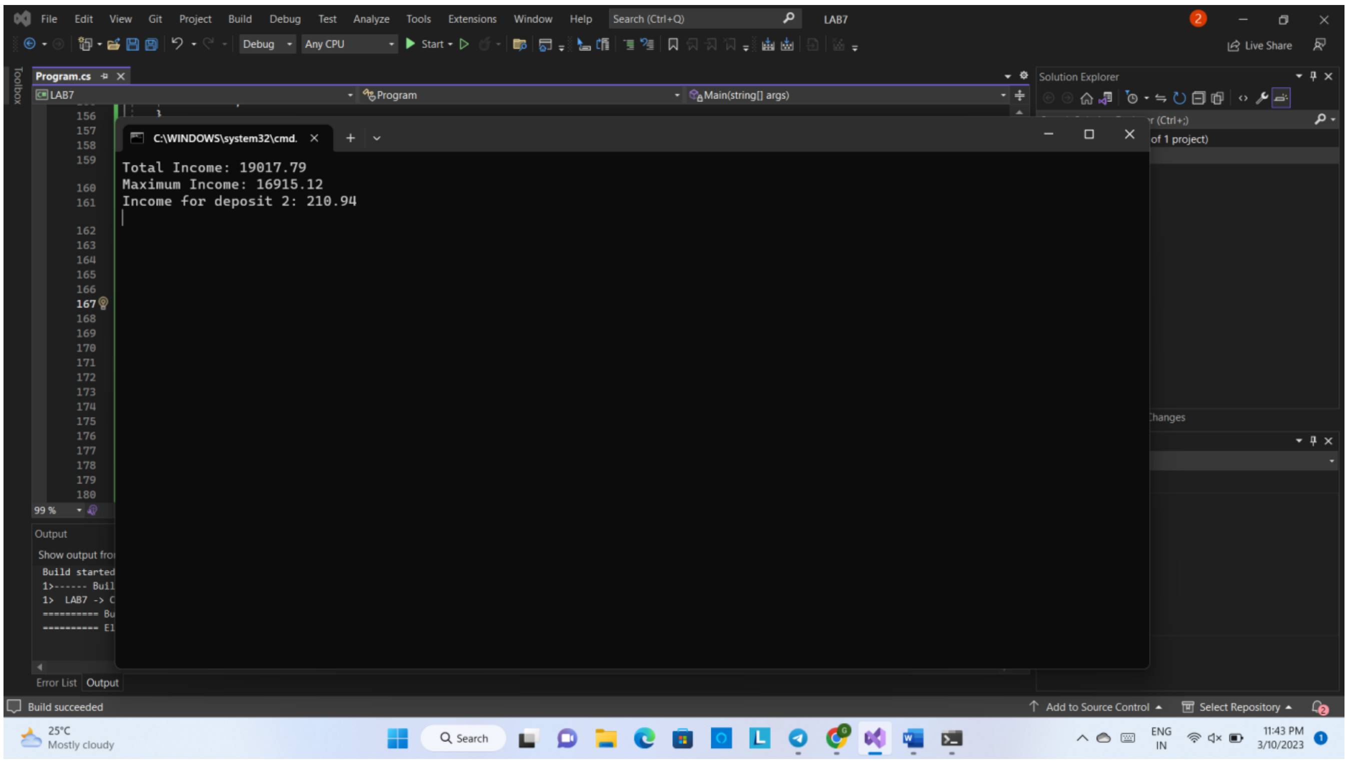
- Private field **deposits** (client deposits) – objects array of type Deposit
- Constructor without parameters, which creates empty array deposits consisting of 10 elements
- Method **AddDeposit** with parameter **deposit** for adding regular, special or long-term account into array on the first empty spot and returning true, or returning false, if accounts number limit is depleted (no empty space in array).
- Method **TotalIncome**, returning total income amount based on all client's deposits upon deposits expiration.
- Method **MaxIncome**, returning maximum deposit income of all client's deposits upon deposits expiration.
- Method **GetIncomeByNumber** with integer parameter **number** (deposit number, which equals its index in array, increased by one), returning income from deposit with such number. If deposit with such number does not exist, method returns 0 value.

Solution:

```
using System;

// Abstract class Deposit
abstract class Deposit
{
    public double Amount { get; }
    public int Period { get; }

    public Deposit(double depositAmount, int depositPeriod)
    {
        Amount = depositAmount;
        Period = depositPeriod;
    }

    public abstract double Income();
}

// Class BaseDeposit
class BaseDeposit : Deposit
{
    public BaseDeposit(double depositAmount, int depositPeriod) : base(depositAmount,
depositPeriod) { }

    public override double Income()
    {
        double sum = Amount;
        double income = 0;

        for (int i = 0; i < Period; i++)
        {
            income += sum * 0.05;
            sum += income;
            income = Math.Round(income, 2);
        }

        return income;
    }
}

// Class SpecialDeposit
class SpecialDeposit : Deposit
{
    public SpecialDeposit(double depositAmount, int depositPeriod) : base(depositAmount,
depositPeriod) { }

    public override double Income()
    {
        double sum = Amount;
        double income = 0;

        for (int i = 1; i <= Period; i++)
        {
            income += sum * i / 100.0;
            sum += income;
            income = Math.Round(income, 2);
        }

        return income;
    }
}

// Class LongDeposit
class LongDeposit : Deposit
{
    public LongDeposit(double depositAmount, int depositPeriod) : base(depositAmount,
depositPeriod) { }

    public override double Income()
    {
        double sum = Amount;
        double income = 0;

        for (int i = 1; i <= Period; i++)
        {
            if (i > 6)
            {
                income += sum * 0.15;
                sum += income;
                income = Math.Round(income, 2);
            }
        }

        return income;
    }
}

// Class Client
class Client
{
    private Deposit[] deposits;

    public Client()
    {
        deposits = new Deposit[10];
```

```csharp
        }

        public bool AddDeposit(Deposit deposit)
        {
            for (int i = 0; i < deposits.Length; i++)
            {
                if (deposits[i] == null)
                {
                    deposits[i] = deposit;
                    return true;
                }
            }

            return false;
        }

        public double TotalIncome()
        {
            double totalIncome = 0;

            foreach (Deposit deposit in deposits)
            {
                if (deposit != null)
                {
                    totalIncome += deposit.Income();
                }
            }

            return totalIncome;
        }

        public double MaxIncome()
        {
            double maxIncome = 0;

            foreach (Deposit deposit in deposits)
            {
                if (deposit != null)
                {
                    double income = deposit.Income();
                    if (income > maxIncome)
                    {
                        maxIncome = income;
                    }
                }
            }

            return maxIncome;
        }

        public double GetIncomeByNumber(int number)
        {
            if (number < 1 || number > deposits.Length)
            {
                return 0;
            }

            Deposit deposit = deposits[number - 1];
            if (deposit != null)
            {
                return deposit.Income();
            }
            return 0;
        }
    }

    // Sample usage
    class Program
    {
        static void Main(string[] args)
        {
            Client client = new Client();
            client.AddDeposit(new BaseDeposit(1000, 3));
            client.AddDeposit(new SpecialDeposit(2000, 4));
            client.AddDeposit(new LongDeposit(3000, 4));
            client.AddDeposit(new BaseDeposit(5000, 2));
            client.AddDeposit(new SpecialDeposit(10000, 1));
            client.AddDeposit(new LongDeposit(20000, 10));
            client.AddDeposit(new BaseDeposit(1500, 6));
            client.AddDeposit(new SpecialDeposit(2500, 2));
            client.AddDeposit(new LongDeposit(5000, 5));
            client.AddDeposit(new BaseDeposit(2000, 4));
            // Calculating the total income for the client
            double totalIncome = client.TotalIncome();
            Console.WriteLine("Total Income: {0}", totalIncome);

            // Calculating the maximum income for the client
            double maxIncome = client.MaxIncome();
            Console.WriteLine("Maximum Income: {0}", maxIncome);

            // Getting the income for a specific deposit
            int depositNumber = 2;
            double incomeByNumber = client.GetIncomeByNumber(depositNumber);
            Console.WriteLine("Income for deposit {0}: {1}", depositNumber, incomeByNumber);

            Console.ReadLine();
        }
    }
```

Output:



```
Total Income: 19017.79
Maximum Income: 16915.12
Income for deposit 2: 210.94
```

TASK 2: To add the following new functionalities to the project created in task Aggregation:

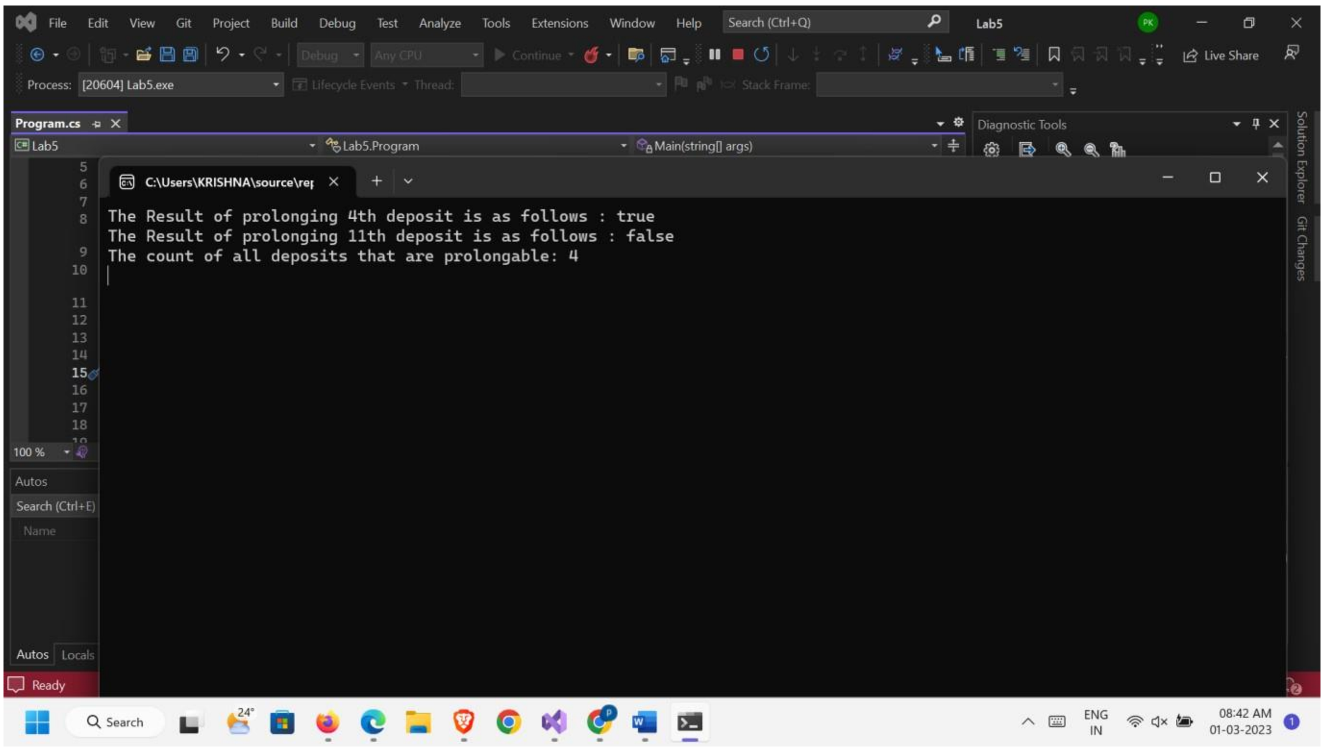
1. To create interface **Iprolongable** (prolonging deposit) and declare within it method **CanToProlong** without parameters that returns logic value true or false, depending on the fact whether this specific deposit can be prolonged or not.
2. To implement interface **IProlongable** in classes **SpecialDeposit** and **LongDeposit**.
3. In addition, special deposit (**SpecialDeposit**) can be prolonged only when more than 1000 UAH were deposited, and long-term deposit (**LongDeposit**) can be prolonged if the period of deposit is no longer than 3 years.
4. To implement standard generic interface **IComparable<Deposit>** in abstract class **Deposit**. Total sum amount (sum deposited plus interest during entire period) should be considered as comparison criteria of **Deposit** instances.
5. To implement additionally in class **Client**:

- interface **IEnumerable<Deposit>**.
- Method **SortDeposits**, which performs deposits sorting in array **deposits** in descending order of total sum amount on deposit upon deposit expiration.
- Method **CountPossibleToProlongDeposit**, which returns integer – amount of current client's deposits that can be prolonged.

Solution:

```
interface Iprolongable
    {
        bool canToProlong();
    }
    abstract class Deposit
    {
        public double amount;
public int period;
        public Deposit(double depositeAmount, int depositePeriod)
        {
            this.amount = depositeAmount;
this.period = depositePeriod;
        }
        public abstract double income();
    }
    class BaseDeposit : Deposit
    {
        public BaseDeposit(double amount, int period)
        {
            base.amount = amount;
base.period = period;
        }
        public override double income()
        {
            return (amount * period) / 20;
        }
}
    class SpecialDeposit : Iprolongable,Deposit
    {
        public SpecialDeposit(double amount, int period)
        {
            base.amount = amount;
base.period = period;
        }
        public override double income()
        {
            return amount * Math.Pow(1 + 1 / 100, period);
        }
        public bool canToProlong()
        {
            if (amount > 1000) return true;
return false;
        }
}
    class LongDeposit : Iprolongable,Deposit
    {
        public bool canToProlong()
        {
            if (period > 3) return true;
return false;
        }
        public LongDeposit(double amount, int peroid)
        {
            base.amount = amount;
base.period = period;
        }
        public override double income()
        {
            return amount * Math.Pow(1 + 15 / 100, period - 6);
        }
}
    public class Client
    {
        private Deposit deposits[] = new Deposit[10];
public Client()
        {
            deposits[] = new Deposit[10];
        }
        IEnumerable<Deposit>;
        public int CountPossibleToProlong()
        {
int c = 0;
            for(int i = 0; i < 10; i++)
            {
                if (deposits[i].canToProlong()) c++;
            }
return c;          }
        public bool AddDeposit(Deposit d)
        {
            for (int i = 0; i < 10; i++)
            {
                if (deposits[i] == null)
                {
                    deposits[i] == d;
return true;
                }
}             return
false;
        }
        public double TotalIncome()
        {
            double sum = 0;
            for (int i = 0; i < 10; i++)
            {
                sum += deposits[i].income();
            }
return sum;
        }
        public double maxIncome()
        {
            double mx = -1;
for (int i = 0; i < 10; i++)
{
                mx = Math.Max(mx, deposits[i].income());
            }
return mx;
        }
        public doublegetIncomebyNumber(int n)
        {
            if (n > 10) return 0;
            return deposits[n - 1].income();
        }
    }
```

Output:



```
The Result of prolonging 4th deposit is as follows : true
The Result of prolonging 11th deposit is as follows : false
The count of all deposits that are prolongable: 4
```

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Lab5
Live Share
Process: [20604] Lab5.exe
Program.cs
Lab5.Program
Main(string[] args)
Diagnostic Tools
C:\Users\KRISHNA\source\re
Autos
Locals
Ready