2100030320

M. BHAVANA

EPAM – LAB-5

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 Income is the difference between sum, withdrawn from deposit upon expiration date and deposited sum.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.

What's New? Program.cs

lab-5-task-1-deposit Program

```
// See https://aka.ms/new-console-template for more information
using System;
12 references
public abstract class Deposit{
    4 references
    public decimal Amount { get; }
    4 references
    public int Period { get; }
    3 references
    public Deposit(decimal depositAmount, int depositPeriod)   {
        Amount = depositAmount;
        Period = depositPeriod;
    }
    6 references
    public abstract decimal Income();
}
2 references
public class BaseDeposit : Deposit{
    1 reference
    public BaseDeposit(decimal amount, int period) : base(amount, period) { }
    4 references
    public override decimal Income()   {
        decimal totalIncome = 0;
        decimal currentAmount = Amount;
        for (int i = 0; i < Period; i++)      {
            decimal monthlyInterest = currentAmount * 0.05m;
            totalIncome += monthlyInterest;
            currentAmount += monthlyInterest;
            currentAmount = Math.Round(currentAmount, 2);     }
        return totalIncome;
    }
}
2 references
public class SpecialDeposit : Deposit{
```

lab-5-task-1-deposit | Program

```
1 reference
public SpecialDeposit(decimal amount, int period) : base(amount, period) { }
4 references
public override decimal Income()  {
    decimal totalIncome = 0;
    decimal currentAmount = Amount;
    for (int i = 1; i <= Period; i++)       {
        decimal monthlyInterest = currentAmount * (i / 100m);
        totalIncome += monthlyInterest;
        currentAmount += monthlyInterest;        }
    return totalIncome;    }
}
2 references
public class LongDeposit : Deposit{
1 reference
public LongDeposit(decimal amount, int period) : base(amount, period) { }
4 references
public override decimal Income()   {
    decimal totalIncome = 0;
    decimal currentAmount = Amount;
    for (int i = 1; i <= Period; i++)       {
        if (i > 6)           {
            decimal monthlyInterest = currentAmount * 0.15m;
            totalIncome += monthlyInterest;
            currentAmount += monthlyInterest;            }        }
    return totalIncome;
}
}
3 references
public class Client{
    private Deposit[] deposits;
1 reference
public Client()   {
    deposits = new Deposit[10];   }
```

What's New? | Program.cs

lab-5-task-1-deposit | Program | Main()

```
3 references
public bool AddDeposit(Deposit deposit)  {
    for (int i = 0; i < deposits.Length; i++)      {
        if (deposits[i] == null)           {
            deposits[i] = deposit;
            return true;           }       }
    return false;
}
1 reference
public decimal TotalIncome()   {
    decimal totalIncome = 0;
    foreach (Deposit deposit in deposits)      {
        if (deposit != null)          {
            totalIncome += deposit.Income();          }
    }
    return totalIncome;
}
1 reference
public decimal MaxIncome()   {
    decimal maxIncome = 0;
    foreach (Deposit deposit in deposits)        {
        if (deposit != null)            {
            maxIncome = Math.Max(maxIncome, deposit.Income());           }
    }
    return maxIncome;
}
1 reference
public decimal GetIncomeByNumber(int number)   {
    if (number >= 1 && number <= deposits.Length && deposits[number - 1] != null)       {
        return deposits[number - 1].Income();      }
    return 0;
}
}
```

```
0 references
class Program
{
    0 references
    static void Main()
    {
        Client client = new Client();
        client.AddDeposit(new BaseDeposit(1000m, 12));
        client.AddDeposit(new SpecialDeposit(1500m, 8));
        client.AddDeposit(new LongDeposit(2000m, 10));
        Console.WriteLine($"Total Income: {client.TotalIncome():C}");
        Console.WriteLine($"Maximum Income: {client.MaxIncome():C}");

        for (int i = 1; i <= 3; i++)
        {
            Console.WriteLine($"Income for Deposit {i}: {client.GetIncomeByNumber(i):C}");
        }
    }
}
```

OUTPUT:

```
Microsoft Visual Studio Debu

Total Income: ? 2,922.92
Maximum Income: ? 1,498.01
Income for Deposit 1: ? 795.85
Income for Deposit 2: ? 629.05
Income for Deposit 3: ? 1,498.01

C:\Users\Bhavana\Documents\EPAM\lab-5-task-1-deposit\bin
th code 0.
To automatically close the console when debugging stops,
le when debugging stops.
Press any key to close this window . . .
```

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.

lab5-task2 | Deposit

```
// See https://aka.ms/new-console-template for more information
using System;
using System.Collections;
using System.Collections.Generic;
1 reference
public interface IComparable<T>{
    2 references
    int CompareTo(T other);
}
3 references
public interface IProlongable{
    3 references
    bool CanToProlong();
}
9 references
public abstract class Deposit : IComparable<Deposit>
{
    protected decimal amount;
    protected decimal interestRate;
    protected int period;
    5 references
    public abstract decimal CalculateTotalAmount();
    2 references
    public int CompareTo(Deposit other)
    {
        decimal thisTotalAmount = CalculateTotalAmount();
        decimal otherTotalAmount = other.CalculateTotalAmount();
        return thisTotalAmount.CompareTo(otherTotalAmount);
    }
}
3 references
public class SpecialDeposit : Deposit, IProlongable
{
    2 references
```

What's New? | Program.cs

lab5-task2 | Deposit

```
    public SpecialDeposit(decimal amount, decimal interestRate, int period)    {
        this.amount = amount;
        this.interestRate = interestRate;
        this.period = period;
    }
    4 references
    public override decimal CalculateTotalAmount()    {
        return amount + (amount * interestRate * period);
    }
    2 references
    public bool CanToProlong()   {
        return amount > 1000;
    }
}
2 references
public class LongDeposit : Deposit, IProlongable{
    1 reference
    public LongDeposit(decimal amount, decimal interestRate, int period)    {
        this.amount = amount;
        this.interestRate = interestRate;
        this.period = period;
    }
    4 references
    public override decimal CalculateTotalAmount()
    {
        return amount + (amount * interestRate * period);
    }
    2 references
    public bool CanToProlong()    {
        return period <= 3;
    }
}
```

What's New? Program.cs

lab5-task2 Deposit

```
53      private List<Deposit> deposits;
        1 reference
54      public Client()  {
55          deposits = new List<Deposit>();
56      }
        3 references
57      public void AddDeposit(Deposit deposit)  {
58          deposits.Add(deposit);
59      }
        1 reference
60      public void SortDeposits()
61      {
62          deposits.Sort((x, y) => y.CompareTo(x));
63      }
        1 reference
64      public int CountPossibleToProlongDeposit()
65      {
66          int count = 0;
67          foreach (var deposit in deposits)        {
68              if (deposit is IProlongable prolongable && prolongable.CanToProlong())
69                  count++;            }       }
70          return count;
71      }
        2 references
72      public IEnumerator<Deposit> GetEnumerator()  {
73          return deposits.GetEnumerator();
74      }
        0 references
75      IEnumerator IEnumerable.GetEnumerator()  {
76          return GetEnumerator();
77      }
78  }
```

What's New? Program.cs

lab5-task2 Deposit period

```
80  class Program
81  {
        0 references
82      static void Main(string[] args)
83      {
84          Client client = new Client();
85          client.AddDeposit(new SpecialDeposit(1500, 0.05m, 1));
86          client.AddDeposit(new LongDeposit(2000, 0.06m, 4));
87          client.AddDeposit(new SpecialDeposit(800, 0.04m, 2));
88
89          Console.WriteLine("Deposits sorted by total amount:");
90          client.SortDeposits();
91          foreach (var deposit in client)
92          {
93              Console.WriteLine($"Total Amount: {deposit.CalculateTotalAmount()}");
94          }
95
96          Console.WriteLine($"Number of Deposits Possible to Prolong: {client.CountPossibleToProlongDeposit()}");
97      }
98  }
99
100
```

Output:

```
Microsoft Visual Studio Debu

Deposits sorted by total amount:
Total Amount: 2480.00
Total Amount: 1575.00
Total Amount: 864.00
Number of Deposits Possible to Prolong: 1

C:\Users\Bhavana\Documents\EPAM\lab5-task2\lab5-task2\bin\Debug\net8.0\lab5-t
.
To automatically close the console when debugging stops, enable Tools->Option
le when debugging stops.
Press any key to close this window . . .
```