LAB-5

Q1)1,2,3 TASKS 2100030723

```
using System;

public abstract class Deposit
{
   public decimal Amount { get; }
   public int Period { get; }

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

   public abstract decimal Income();
}

public class BaseDeposit : Deposit
{
   public BaseDeposit(decimal amount, int period) : base(amount, period)
   {
   }

   public override decimal Income()
   {
      decimal currentAmount = Amount;
      decimal income = 0;

      for (int i = 0; i < Period; i++)
      {
```

```
        decimal monthlyInterest = currentAmount * 0.05m;
        income += monthlyInterest;
        currentAmount += monthlyInterest;
      }

      return Math.Round(income, 2);
    }
  }

  public class SpecialDeposit : Deposit
  {
    public SpecialDeposit(decimal amount, int period) : base(amount, period)
    {
    }

    public override decimal Income()
    {
      decimal currentAmount = Amount;
      decimal income = 0;

      for (int i = 0; i < Period; i++)
      {
        decimal monthlyInterest = currentAmount * (i + 1) / 100m;
        income += monthlyInterest;
        currentAmount += monthlyInterest;
      }

      return Math.Round(income, 2);
    }
  }

  public class LongDeposit : Deposit
  {
```

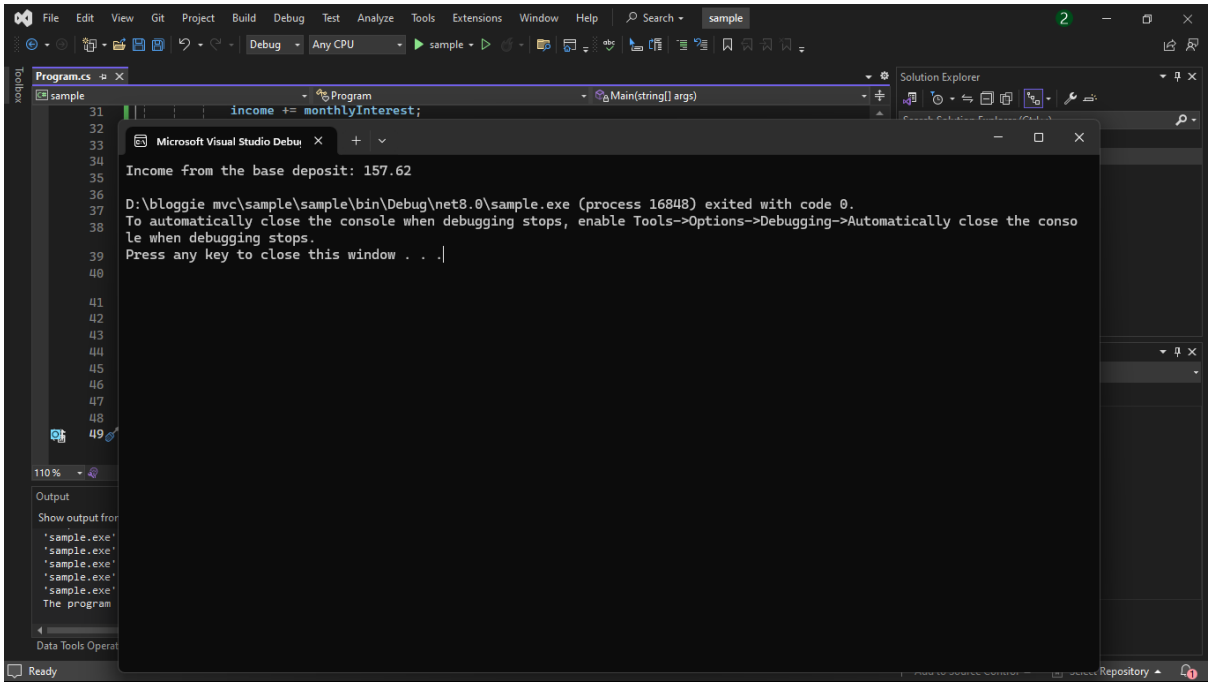
```
    public LongDeposit(decimal amount, int period) : base(amount, period)
    {
    }

    public override decimal Income()
    {
        decimal income = 0;

        for (int i = 0; i < Period; i++)
        {
            if (i >= 6)
            {
                decimal monthlyInterest = Amount * 0.15m;
                income += monthlyInterest;
                Amount += monthlyInterest;
            }
        }

        return Math.Round(income, 2);
    }
}

public class Client
{
    private Deposit[] deposits;

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

    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 decimal TotalIncome()
{
    decimal totalIncome = 0;
    foreach (var deposit in deposits)
    {
        if (deposit != null)
        {
            totalIncome += deposit.Income();
        }
    }
    return Math.Round(totalIncome, 2);
}

public decimal MaxIncome()
{
    decimal maxIncome = 0;
    foreach (var deposit in deposits)
    {
        if (deposit != null)
        {
            decimal income = deposit.Income();
            if (income > maxIncome)
```

```
            {
                maxIncome = income;
            }
        }
    }
    return Math.Round(maxIncome, 2);
  }

  public decimal GetIncomeByNumber(int number)
  {
    if (number > 0 && number <= deposits.Length && deposits[number - 1] != null)
    {
      return Math.Round(deposits[number - 1].Income(), 2);
    }
    return 0;
  }
}

class Program
{
  static void Main(string[] args)
  {
    // Testing the classes
    Client client = new Client();

    BaseDeposit baseDeposit = new BaseDeposit(1000, 3);
    SpecialDeposit specialDeposit = new SpecialDeposit(1000, 3);
    LongDeposit longDeposit = new LongDeposit(1000, 12);

    client.AddDeposit(baseDeposit);
    client.AddDeposit(specialDeposit);
    client.AddDeposit(longDeposit);
```

```
        Console.WriteLine($"Total income: {client.TotalIncome()}");
        Console.WriteLine($"Max income: {client.MaxIncome()}");
        Console.WriteLine($"Income from deposit number 2: {client.GetIncomeByNumber(2)}");
    }
}
```

Output :



Q2) TASK1,2,3

```
using System;
using System.Collections;
using System.Collections.Generic;

public interface Iprolongable
{
```

```
    bool CanToProlong();
}

public abstract class Deposit : IComparable<Deposit>
{
    public decimal Amount { get; }
    public int Period { get; }

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

    public abstract decimal Income();

    public decimal TotalSum()
    {
        return Amount + Income();
    }

    public int CompareTo(Deposit other)
    {
        return TotalSum().CompareTo(other.TotalSum());
    }
}

public class BaseDeposit : Deposit
{
```

```
    public BaseDeposit(decimal amount, int period) : base(amount, period)
    {
    }

    public override decimal Income()
    {
        decimal currentAmount = Amount;
        decimal income = 0;

        for (int i = 0; i < Period; i++)
        {
            decimal monthlyInterest = currentAmount * 0.05m;
            income += monthlyInterest;
            currentAmount += monthlyInterest;
        }

        return Math.Round(income, 2);
    }
}

public class SpecialDeposit : Deposit, Iprolongable
{
    public SpecialDeposit(decimal amount, int period) : base(amount, period)
    {
    }

    public override decimal Income()
    {
        decimal currentAmount = Amount;
```

```
        decimal income = 0;

        for (int i = 0; i < Period; i++)
        {
            decimal monthlyInterest = currentAmount * (i + 1) / 100m;
            income += monthlyInterest;
            currentAmount += monthlyInterest;
        }

        return Math.Round(income, 2);
    }

    public bool CanToProlong()
    {
        return Amount > 1000;
    }
}

public class LongDeposit : Deposit, Iprolongable
{
    public LongDeposit(decimal amount, int period) : base(amount, period)
    {
    }

    public override decimal Income()
    {
        decimal income = 0;

        for (int i = 0; i < Period; i++)
```

```
        {
            if (i >= 6)
            {
                decimal monthlyInterest = Amount * 0.15m;
                income += monthlyInterest;
                Amount += monthlyInterest;
            }
        }

        return Math.Round(income, 2);
    }

    public bool CanToProlong()
    {
        return Period <= 36; // 3 years in months
    }
}

public class Client : IEnumerable<Deposit>
{
    private Deposit[] deposits;
    private int count;

    public Client()
    {
        deposits = new Deposit[10];
        count = 0;
    }
```

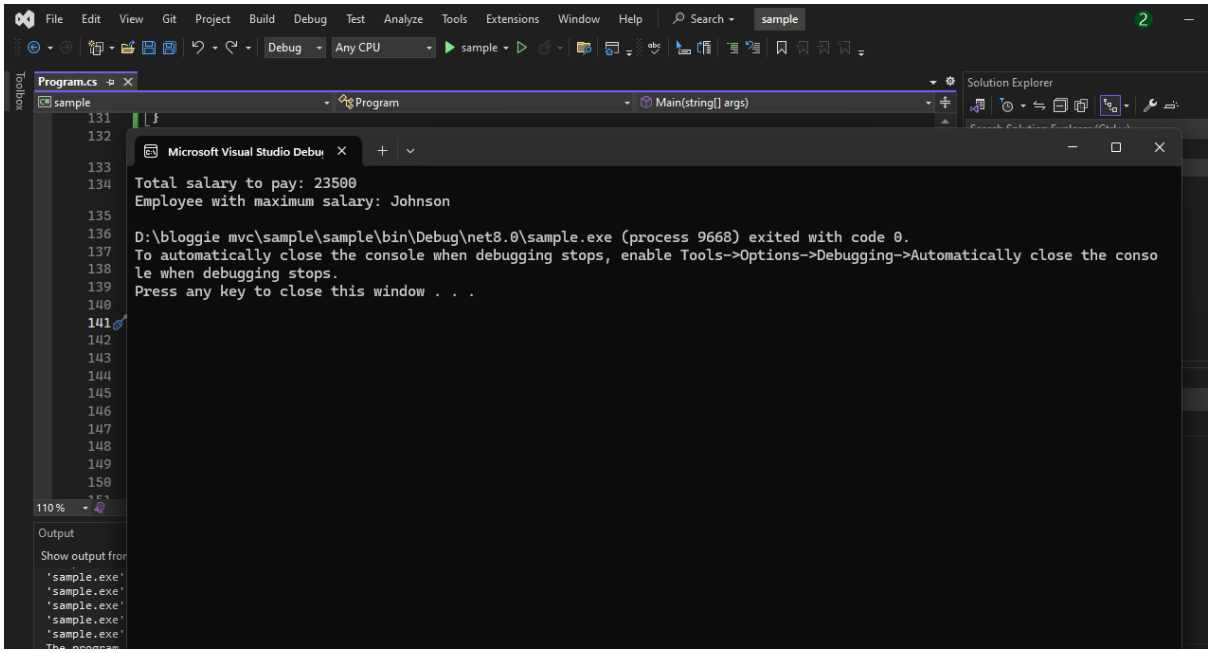
```
public bool AddDeposit(Deposit deposit)
{
   if (count < deposits.Length)
   {
      deposits[count] = deposit;
      count++;
      return true;
   }
   return false;
}

public decimal TotalIncome()
{
   decimal totalIncome = 0;
   foreach (var deposit in deposits)
   {
      if (deposit != null)
      {
         totalIncome += deposit.Income();
      }
   }
   return Math.Round(totalIncome, 2);
}

public decimal MaxIncome()
{
   decimal maxIncome = 0;
   foreach (var deposit in deposits)
   {
```

```
        if (deposit != null)
        {
            decimal income = deposit.Income();
            if (income > maxIncome)
            {
                maxIncome = income;
            }
        }
    }
    return Math.Round(maxIncome, 2);
}

public decimal GetIncomeByNumber(int number)
{
    if (number > 0 && number <= count && deposits[number - 1] != null)
    {
        return Math.Round(deposits[number - 1].Income(), 2);
    }
    return 0;
}

public IEnumerator<Deposit> GetEnumerator()
{
    foreach (var deposit in deposits)
    {
        yield return deposit;
    }
}
```

```
    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }

    public void SortDeposits()
    {
        Array.Sort(deposits, 0, count);
        Array.Reverse(deposits, 0, count);
    }

    public int CountPossibleToProlongDeposit()
    {
        int countPossible = 0;
        foreach (var deposit in deposits)
        {
            if (deposit != null && deposit is Iprolongable &&
((Iprolongable)deposit).CanToProlong())
            {
                countPossible++;
            }
        }
        return countPossible;
    }
}

class Program
{
    static void Main(string[] args)
    {
```

```
    Client client = new Client();

    BaseDeposit baseDeposit = new BaseDeposit(1000, 3);
    SpecialDeposit specialDeposit = new SpecialDeposit(1500, 3);
    LongDeposit longDeposit = new LongDeposit(2000, 24);

    client.AddDeposit(baseDeposit);
    client.AddDeposit(specialDeposit);
    client.AddDeposit(longDeposit);

    Console.WriteLine($"Total income: {client.TotalIncome()}");
    Console.WriteLine($"Max income: {client.MaxIncome()}");
    Console.WriteLine($"Income from deposit number 2:
{client.GetIncomeByNumber(2)}");

    client.SortDeposits();

    Console.WriteLine("\nDeposits after sorting:");
    foreach (var deposit in client)
    {
      Console.WriteLine($"Total sum amount: {deposit.TotalSum()}");
    }

    Console.WriteLine($"Count of possible to prolong deposits:
{client.CountPossibleToProlongDeposit()}");
  }
}
```

Output :


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
sample
Debug
Any CPU
Program.cs
sample
Program
Main(string[] args)
Solution Explorer
Microsoft Visual Studio Debu
Total salary to pay: 23500
Employee with maximum salary: Johnson
D:\bloggie mvc\sample\sample\bin\Debug\net8.0\sample.exe (process 9668) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
Output
Show output fro