Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина «Программирование»

**ОТЧЕТ**

к лабораторной работе №4

на тему:

**«Конструкторы. Статические члены класса.**

**Шаблон проектирования Singleton.»**

БГУИР 6-05-0612-02 124

|  |
| --- |
| Выполнил студент группы 453504  ЯРЦЕВ Александр Александрович |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  РОМАНЮК Максим Валерьевич |
|  |
| (дата, подпись преподавателя) |

Минск 2025

# 1 Индивидуальное задание

**Задание 1. Вариант 5.** Предметная область: Банк - Вклад. В классе хранить информацию о наименовании банка, число вкладов, размер вклада (одинаковый для всех вкладчиков - класс Вклад), размер процентной ставки. Реализовать метод для подсчета общей выплаты по процентам. Реализовать возможность изменения (увеличения и уменьшения) размера вклада.

# 2 Выполнение работы

Для выполнения задания в проект была добавлена папка Classes, в которой находятся классы Bank и Deposit (cм. рисунок 1).

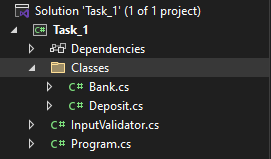


Рисунок 1 – Обозреватель решений

Сперва был реализован класс Bank, в котором имеется свойство Name, что отвечает за наименование банка. Для Name был реализованы метод GetInstance, который соответствует шаблону проектирования Singleton и он же использует оператор присваивания с нулевым объединением (??=) для присвоения нового экземпляра Bank переменной instance, если она в данный момент равна null. Также в классе реализованы и другие методы.

public class Bank

{

private static Bank instance;

private string name;

private List<Deposit> deposits;

private Bank(string name)

{

this.name = name;

deposits = new List<Deposit>();

}

public static Bank GetInstance(string name)

{

if (string.IsNullOrWhiteSpace(name))

throw new ArgumentException("Название банка не может быть пустым.");

return instance ??= new Bank(name);

}

public string Name

{

get { return name; }

set

{

if (string.IsNullOrWhiteSpace(value))

throw new ArgumentException("Название банка не может быть пустым.");

name = value;

}

}

public void AddDeposit(Deposit deposit)

{

deposits.Add(deposit);

}

public decimal CalculateTotalInterest(out List<(decimal Rate, decimal Interest)> details)

{

decimal total = 0;

details = new List<(decimal Rate, decimal Interest)>();

for (int i = 0; i < deposits.Count; i++)

{

decimal interest = deposits[i].CalculateInterest();

total += interest;

details.Add((deposits[i].InterestRate, interest));

}

return total;

}

public int NumberOfDeposits

{

get { return deposits.Count; }

}

}

Рассмотрим реализацию класса Deposit, в котором мы реализуем управление суммой вклада (уменьшение и увелечение общего вклада), установка процентной ставки и расчёт начисленных процентов по вкладам.

Реализацию смотреть ниже:

public class Deposit

{

private static decimal depositAmount; // Размер вклада (общий для всех)

private decimal interestRate; // Процентная ставка

public Deposit(decimal interestRate)

{

this.interestRate = interestRate;

}

public decimal InterestRate

{

get { return interestRate; }

set { interestRate = value; }

}

public static decimal DepositAmount

{

get { return depositAmount; }

set

{

if (value < 0)

throw new ArgumentException("Размер вклада не может быть отрицательным.");

depositAmount = value;

}

}

public static void IncreaseDepositAmount(decimal amount)

{

if (amount < 0)

throw new ArgumentException("Сумма увеличения не может быть отрицательной.");

DepositAmount = depositAmount + amount;

}

public static void DecreaseDepositAmount(decimal amount)

{

if (amount < 0)

throw new ArgumentException("Сумма уменьшения не может быть отрицательной.");

DepositAmount = depositAmount - amount;

}

public decimal CalculateInterest()

{

return depositAmount \* (interestRate / 100);

}

}

Покажем принцип работы этой программы в классе Program, где, помимо всего, используются методы GetValidString и GetValidDecimal из класса InputValidator в качестве проверки на ввод для типов данных string и decimal. Используя шаблон проектирования Singleton мы создаём объект класса Bank и реализуем принцип работы программы используя оператор выбора switch.

class Program

{

static void Main(string[] args)

{

string bankName = InputValidator.GetValidString("Введите наименование банка: ");

Bank bank = Bank.GetInstance(bankName);

decimal initialDeposit = InputValidator.GetValidDecimal("Введите размер вклада: ");

Deposit.DepositAmount = initialDeposit;

while (true)

{

Console.WriteLine("\nМеню:");

Console.WriteLine("1. Добавить вклад");

Console.WriteLine("2. Увеличить размер вклада");

Console.WriteLine("3. Уменьшить размер вклада");

Console.WriteLine("4. Вывести информацию о банке");

Console.WriteLine("5. Выход");

Console.Write("Выберите действие: ");

string choice = Console.ReadLine();

if (string.IsNullOrWhiteSpace(choice))

{

Console.WriteLine("Ошибка: Выбор не может быть пустым.");

continue;

}

Console.WriteLine();

switch (choice)

{

case "1":

decimal rate = InputValidator.GetValidDecimal("Введите процентную ставку: ");

Deposit deposit = new Deposit(rate);

bank.AddDeposit(deposit);

Console.WriteLine("Вклад добавлен.");

break;

case "2":

decimal increaseAmount = InputValidator.GetValidDecimal("Введите сумму для увеличения: ");

try

{

Deposit.IncreaseDepositAmount(increaseAmount);

Console.WriteLine("Размер вклада увеличен.");

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

break;

case "3":

decimal decreaseAmount = InputValidator.GetValidDecimal("Введите сумму для уменьшения: ");

try

{

Deposit.DecreaseDepositAmount(decreaseAmount);

Console.WriteLine("Размер вклада уменьшен.");

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

break;

case "4":

Console.WriteLine($"Наименование банка: {bank.Name}");

Console.WriteLine($"Число вкладов: {bank.NumberOfDeposits}");

Console.WriteLine($"Текущий размер вклада: {Deposit.DepositAmount}");

var totalInterest = bank.CalculateTotalInterest(out var details);

Console.WriteLine("Детали выплат по процентам:");

for (int i = 0; i < details.Count; i++)

{

Console.WriteLine($"Вклад {i + 1}: Ставка {details[i].Rate}%, Проценты {details[i].Interest}");

}

Console.WriteLine($"Общая выплата по процентам: {totalInterest}");

break;

case "5":

Console.WriteLine("Выход из программы.");

return;

default:

Console.WriteLine("Ошибка: Неверный выбор. Попробуйте снова.");

break;

}

}

}

}

Результат работы программы продемонстрирован ниже (см. рисунок 2).

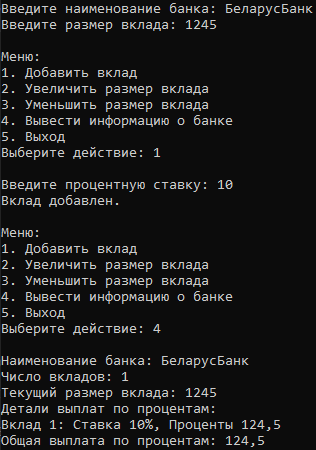


Рисунок 2 – Результат работы программы

# Вывод

В ходе лабораторной работы были изучены назначения конструкторов в классе, валидация ввода через класс InputValidator, возможности членов класса с модификатором static, шаблон проектирования Singleton для гарантии единственного экземпляра банка.