|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет социально-экономических и компьютерных наук* |
|  |
| Гуцол Степан Дмитриевич  **ЛАБОРАТОРНАЯ РАБОТА №9**  по направлению подготовки *38.03.05 Бизнес-информатика*  образовательная программа «Разработка информационных систем для бизнеса»   |  |  | | --- | --- | |  | Руководитель  Преподаватель кафедры ИТБ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Найданов И.В. |   Пермь, 2022 |

**Оглавление**

[**1. Постановка задачи** 3](#_Toc126695754)

[**3. Диаграмма классов** 5](#_Toc126695755)

[**4. Код программы(листинг)** 6](#_Toc126695756)

[**5. Код Unit-Тестов** 7](#_Toc126695757)

[**6. Анализ покрытия кода тестами** 11](#_Toc126695758)

[ПРИЛОЖЕНИЕ А **Программа program.cs** 12](#_Toc126695759)

[ПРИЛОЖЕНИЕ Б **Программа Money.cs** 20](#_Toc126695760)

[ПРИЛОЖЕНИЕ В **Программа MoneyArray.cs** 23](#_Toc126695761)

**1. Постановка задачи**

**Часть 1:**

1. Реализовать (в отдельном файле) определение нового класса/типа данных (закрытые атрибуты, свойства, конструкторы, инициализация и вывод атрибутов).

2. Для демонстрации работы с объектами написать главную функцию, в которой создаются объекты класса и выводится информация, которая содержится в атрибутах.

3. Написать функцию, реализующую указанное в варианте действие. Рассмотреть два варианта:

1) статическую функцию;

2) метод класса.

В основной функции продемонстрировать работу функции.

4. Используя статическую компоненту класса подсчитать количество созданных в программе объектов.

**Описание 8 варианта:**

Класс – Money, поля – int rubles, int kopeks, метод – Вычитание переменной типа Money (учесть, что рублей и копеек не может быть меньше 0). Результат должен быть типа Money.

**Часть 2:**

1. Добавить к реализованному классу указанные в варианте перегруженные операции.

2. Написать демонстрационную программу, в которой создаются объекты пользовательских классов и выполняются указанные операции.

**Описание 8 варианта:**

Унарные операции:

++ добавление копейки к объекту типа Money (учесть, что копеек не может быть больше 99).

-- вычитание копейки из объекта типа Money (учесть, что копеек и рублей не может быть меньше 0).

Операции приведения типа:

int (неявная) – результатом является количество рублей (копейки отбрасываются);

double (явная) – результатом является копейки, рубли отбрасываются, результат <1.

Бинарные операции:

- Money m, целое число (лево- и право- сторонние операции), результат должен быть типа Money.

- Money m, Money m, результат должен быть типа Money.

Результат не может быть отрицательным.

**Часть 3:**

1. Реализовать класс-коллекцию (в отдельном файле), полем которого является одномерный массив (не использовать стандартные коллекции C#!) из элементов заданного в варианте типа.

В классе реализовать

• конструктор без параметров,

• конструктор с параметрами, заполняющий элементы случайными значениями,

• конструктор с параметрами, позволяющий заполнить массив элементами, заданными пользователем с клавиатуры,

• метод для просмотра элементов массива.

2. Реализовать индексатор для доступа к элементам коллекции. Предусмотреть проверку при выходе индекса за пределы массива.

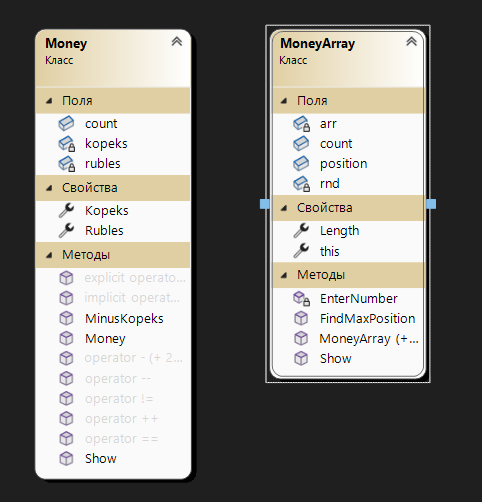
3. Написать демонстрационную программу, позволяющую создать массив разными способами и распечатать элементы массива. Подсчитать количество созданных объектов.

4. Написать функцию в классе Program для выполнения указанного в варианте задания (использовать индексатор и, если необходимо, перегрузить нужные для выполнения задачи операции).

**Описание 8 варианта:**

Максимальное значение

**3. Диаграмма классов**



**Рисунок 1 – Диаграмма классов**

**4. Код программы(листинг)**

Так как код программы получился достаточно объёмным, то его можно найти в приложении, также в приложении можно найти код классов Money и MoneyArray.

**5. Код Unit-Тестов**

using Labo9;

using System.Data.SqlTypes;

namespace TestProject1

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestMethodInit()

{

Money expected = new Money(5, 50);

Money actual = new Money(4, 150);

bool expectedTest = true;

bool actualTest = actual == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodInit2()

{

Money expected = new Money(1, 50);

Money actual = new Money(-2, 150);

bool expectedTest = true;

bool actualTest = actual == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodMinusKopeks()

{

Money expected = new Money(4, 50);

Money actual = new Money(5 , 0);

Money test = actual.MinusKopeks(50);

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodMinusKopeks2()

{

Money expected = new Money(0, 0);

Money actual = new Money(5, 0);

Money test = actual.MinusKopeks(501);

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodIncrement()

{

Money expected = new Money(4, 50);

Money actual = new Money(4, 49);

Money test = actual++;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodToInt()

{

int expected = 4;

Money actual = new Money (4,50);

int test = actual;

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestMethodToDouble()

{

int expected = 50;

Money actual = new Money(4,50);

double test = (double)actual;

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestMethodKopeksMinusWallet()

{

Money expected = new Money(2, 13);

Money walletMinus = new Money(1, 36);

int kopeksActual = 349;

Money test = kopeksActual - walletMinus;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodKopeksMinusWallet2()

{

Money expected = new Money(0, 0);

Money walletMinus = new Money(4, 36);

int kopeksActual = 349;

Money test = kopeksActual - walletMinus;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodWalletMinusKopeks()

{

Money expected = new Money(2, 13);

Money walletMinus = new Money(3, 56);

int kopeksActual = 143;

Money test = walletMinus - kopeksActual;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodDecrement()

{

Money expected = new Money(3, 1);

Money actual = new Money(3, 2);

Money test = actual--; ;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodDecrement2()

{

Money expected = new Money(2, 99);

Money actual = new Money(3, 0);

Money test = --actual; ;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodDecrement3()

{

Money expected = new Money(0, 0);

Money actual = new Money(0, 0);

Money test = --actual; ;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodWalletMinusWallet()

{

Money expected = new Money(5, 25);

Money actual = new Money(7, 52);

Money walletMinus = new Money(2, 27);

Money test = actual- walletMinus;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodWalletMinusWallet2()

{

Money expected = new Money(0, 0);

Money actual = new Money(7, 52);

Money walletMinus = new Money(8, 0);

Money test = actual - walletMinus;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodGetLength()

{

int expected = 3;

MoneyArray []arr = new MoneyArray[3];

int test = arr.Length;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodWalletMinusKopeks2()

{

Money expected = new Money(0,0);

Money walletMinus = new Money(3, 56);

int kopeksActual = 400;

Money test = walletMinus - kopeksActual;

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodWalletNotEquals()

{

Money walletOne = new Money(-3, 56);

Money walletTwo = new Money(3, 56);

bool expected = true;

bool test = (walletOne!=walletTwo);

bool expectedTest = true;

bool actualTest = test == expected;

Assert.AreEqual(expectedTest, actualTest);

}

[TestMethod]

public void TestMethodEnterNumber()

{

MoneyArray[] arr = new MoneyArray[3];

MoneyArray expected = arr[2];

bool expectedTest = true;

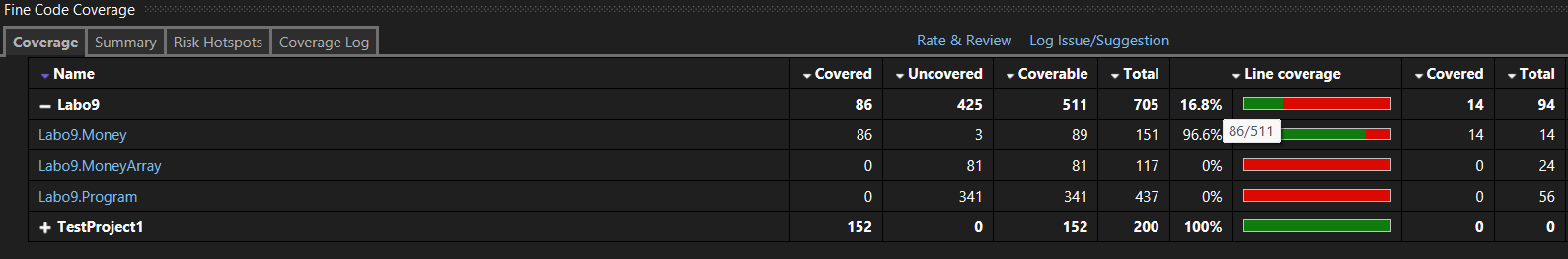
bool actualTest = arr[2] == expected;

Assert.AreEqual(expectedTest, actualTest);

}

}

**6. Анализ покрытия кода тестами**

******

**Рисунок 2 – Анализ покрытия кода тестами**

# ПРИЛОЖЕНИЕ А **Программа program.cs**

using System.Diagnostics.Metrics;

using System.Runtime.Versioning;

using System.Security.Cryptography.X509Certificates;

namespace Labo9

// Вариант - 8

{

internal class Program

{

static void Main(string[] args)

{

int enter = 0; // Переменная ввода данных

bool exitProgramm = false;

bool exitTaskOne = false;

bool exitTaskTwo = false;

bool exitTaskTwoMenuFifth = false;

bool exitTaskTwoMenuSixth = false;

bool exitTaskThree = false;

bool exitTaskThreeMenu = false;

do

{

Console.WriteLine(@"Меню:

[1] - 1 часть

[2] - 2 часть

[3] - 3 часть

[0] - Выход из программы

Введите цифру необходимого пункта меню:");

enter = EnterNumber();

switch (enter)

{

case 1: // Первая часть

do

{

Console.Clear();

Console.WriteLine(@"Меню:

[1] - Создать объект типа Money и найти разность.

[2] - Вывести на экран количество созданных объектов.

[0] - Выход из программы

Введите цифру необходимого пункта меню:");

enter = EnterNumber();

switch (enter)

{

case 1:

Console.Clear();

Console.WriteLine("Введите количество рублей для объекта:");

int rublesOne = EnterNumber();

Console.WriteLine("Введите количество копеек для объекта:");

int kopeksOne = EnterNumber();

Money walletOne = new Money(rublesOne, kopeksOne);

Console.Write("Ваш объект : ");

walletOne.Show();

Console.WriteLine("Введите количество копеек для вычитания из объекта");

int kopeksMinus = EnterNumber();

Console.WriteLine("Введите количество рублей для вычитания из объекта:");

int rublesMinus = EnterNumber();

kopeksMinus += rublesMinus \* 100;

Money walletMinus = walletOne.MinusKopeks(kopeksMinus);

Console.Write("Ваш объект разности: ");

walletMinus.Show();

Console.WriteLine("[0] - Вернуться назад");

ReturnBack(enter);

break;

case 2:

Console.Clear();

Console.WriteLine($"Количество созданных объектов: {Money.count}");

Console.WriteLine("[0] - Вернуться назад");

ReturnBack(enter);

break;

case 0:

Console.Clear();

exitTaskOne = true;

break;

}

} while (!exitTaskOne);

break;

case 2:

do

{

Console.Clear();

Console.WriteLine(@"Меню:

Унарные операции:

[1] - ++ добавление копейки к объекту типа Money

[2] - -- вычитание копейки из объекта типа Money

Операции приведения типа:

[3] - int (неявная) – результатом является количество рублей

[4] - double (явная) – результатом является копейки

Бинарные операции:

[5] - Разность объекта типа Money и целого числа

[6] - разность двух объектов типа Money

[0] - Выход из программы

Введите цифру необходимого пункта меню:");

enter = EnterNumber();

switch (enter)

{

case 1:

Console.Clear();

Console.WriteLine("Введите количество рублей для объекта:");

int rublesTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для объекта:");

int kopeksTwo = EnterNumber();

Money walletTwo = new Money(rublesTwo, kopeksTwo);

Console.Write("Ваш объект : ");

walletTwo.Show();

walletTwo++;

Console.Write("Ваш объект после операции: ");

walletTwo.Show();

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 2:

Console.Clear();

Console.WriteLine("Введите количество рублей для объекта:");

rublesTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для объекта:");

kopeksTwo = EnterNumber();

walletTwo = new Money(rublesTwo, kopeksTwo);

Console.Write("Ваш объект : ");

walletTwo.Show();

walletTwo--;

Console.Write("Ваш объект после операции: ");

walletTwo.Show();

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 3:

Console.Clear();

Console.WriteLine("Введите количество рублей для объекта:");

rublesTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для объекта:");

kopeksTwo = EnterNumber();

walletTwo = new Money(rublesTwo, kopeksTwo);

Console.Write("Ваш объект : ");

walletTwo.Show();

int rublesImplicit = walletTwo;

Console.Write("Ваш объект после приведения: ");

Console.WriteLine(rublesImplicit);

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 4:

Console.Clear();

Console.WriteLine("Введите количество рублей для объекта:");

rublesTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для объекта:");

kopeksTwo = EnterNumber();

walletTwo = new Money(rublesTwo, kopeksTwo);

Console.Write("Ваш объект : ");

walletTwo.Show();

double kopeksExplicit = (double)walletTwo;

Console.Write("Ваш объект после приведения: ");

Console.WriteLine(kopeksExplicit / 100);

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 5:

do {

Console.Clear();

Console.WriteLine(@"Меню:

[1] - Объект - копейки

[2] - Копейки - объект

[0] - Вернуться назад");

enter = EnterNumber();

switch (enter)

{

case 1:

Console.Clear();

Console.WriteLine("Введите количество рублей для объекта:");

rublesTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для объекта:");

kopeksTwo = EnterNumber();

walletTwo = new Money(rublesTwo, kopeksTwo);

Console.Write("Ваш объект : ");

walletTwo.Show();

Console.WriteLine("Введите количество копеек для вычитания из объекта");

int kopeksMinus = EnterNumber();

walletTwo = walletTwo - kopeksMinus;

Console.Write("Ваш объект разности: ");

walletTwo.Show();

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 2:

Console.Clear();

Console.WriteLine("Введите количество рублей для объекта:");

rublesTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для объекта:");

kopeksTwo = EnterNumber();

walletTwo = new Money(rublesTwo, kopeksTwo);

Console.Write("Ваш объект : ");

walletTwo.Show();

Console.WriteLine("Введите количество копеек, из которых будет сделан вычет");

kopeksMinus = EnterNumber();

walletTwo = kopeksMinus - walletTwo;

Console.Write("Ваш объект разности: ");

walletTwo.Show();

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 0:

exitTaskTwoMenuFifth = true;

break;

}

} while (!exitTaskTwoMenuFifth);

break;

case 6:

Console.Clear();

Console.WriteLine("Введите количество рублей для первого объекта:");

rublesTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для первого объекта:");

kopeksTwo = EnterNumber();

walletTwo = new Money(rublesTwo, kopeksTwo);

Console.Write("Ваш первый объект : ");

walletTwo.Show();

Console.WriteLine("Введите количество рублей для второго объекта:");

int rublesMinusTwo = EnterNumber();

Console.WriteLine("Введите количество копеек для второго объекта:");

int kopeksMinusTwo = EnterNumber();

Money walletMinus = new Money(rublesMinusTwo, kopeksMinusTwo);

Console.Write("Ваш второй объект : ");

walletMinus.Show();

walletTwo = walletTwo - walletMinus;

Console.Write("Ваш объект разности: ");

walletTwo.Show();

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 0:

Console.Clear();

exitTaskTwo = true;

break;

}

} while (!exitTaskTwo);

break;

case 3:

do

{

Console.Clear();

Console.WriteLine(@"Меню:

[1] - Создать массив с помощью ДСЧ

[2] - Создать массив ручным вводом

[3] - Распечатать количество созданных элементов

[0] - Вернуться назад");

enter = EnterNumber();

switch (enter) {

case 1:

Console.Clear();

Console.WriteLine("Введите необходимое количество элементов массива:");

int arrayLength = EnterNumber();

MoneyArray moneyArray = new MoneyArray(arrayLength);

moneyArray.Show();

Console.Write(@"

Максимальное значение - ");

moneyArray[MoneyArray.FindMaxPosition(moneyArray)].Show();

Console.WriteLine(@"

[1] - Распечатать элемент массива под определённым номером

[0] - Вернуться назад");

enter = EnterNumber();

do

{

switch (enter)

{

case 1:

bool isCorrectPosition = true;

do

{

isCorrectPosition = true;

Console.WriteLine("Введите номер нужного элемента массива:");

int elementPosition = EnterNumber() - 1;

try

{

moneyArray[elementPosition].Show();

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Вы неправильно задали номер элемента:");

isCorrectPosition = false;

}

}while (!isCorrectPosition);

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 0:

exitTaskThreeMenu = true;

break;

}

break;

} while (!exitTaskThreeMenu);

break;

case 2:

Console.Clear();

Console.WriteLine("Введите необходимое количество элементов массива:");

double arrayLengthDouble = EnterNumber();

moneyArray = new MoneyArray(arrayLengthDouble);

Console.Clear();

moneyArray.Show();

Console.Write(@"

Максимальное значение - ");

moneyArray[MoneyArray.FindMaxPosition(moneyArray)].Show();

Console.WriteLine(@"

[1] - Распечатать элемент массива под определённым номером

[0] - Вернуться назад");

enter = EnterNumber();

do

{

switch (enter)

{

case 1:

bool isCorrectPosition = true;

do

{

isCorrectPosition = true;

Console.WriteLine("Введите номер нужного элемента массива:");

int elementPosition = EnterNumber() - 1;

try

{

moneyArray[elementPosition].Show();

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Вы неправильно задали номер элемента:");

isCorrectPosition = false;

}

} while (!isCorrectPosition);

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 0:

exitTaskThreeMenu = true;

break;

}

break;

} while (!exitTaskThreeMenu);

break;

case 3:

Console.Clear();

Console.WriteLine($@"Количество созданных объектов: {MoneyArray.count}");

Console.WriteLine(@"

[0] - Вернуться назад");

ReturnBack(enter);

break;

case 0:

exitTaskThree = true;

break;

}

} while (!exitTaskThree);

break;

case 0:

exitProgramm = true;

break;

}

Console.Clear();

} while (!exitProgramm);

}

static int EnterNumber() // Ввод целого числа

{

bool ok;

int Num;

do

{

ok = int.TryParse(Console.ReadLine(), out Num);

if (!ok)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Вы ввели неверное значение!");

Console.ForegroundColor = ConsoleColor.White;

}

} while (!ok);

if (Num <= 0)

{

return 0;

}

else

{

return Num;

}

}

static void ReturnBack(int enter)

{

do

{

enter = EnterNumber();

} while (enter != 0);

}

}

}

# ПРИЛОЖЕНИЕ Б **Программа Money.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace Labo9

{

public class Money

{

public static int count = 0;

int rubles;

int kopeks;

public int Rubles

{

get

{

return rubles;

}

set

{

if (value < 0)

{

rubles = 0;

}

else

{

rubles = value;

}

}

}

public int Kopeks

{

get => kopeks;

set

{

if (value < 0)

{

kopeks = 0;

}

else

{

if (value < 100)

{

kopeks = value;

}

else

{

rubles += value / 100;

kopeks = value % 100;

}

}

}

}

public Money(int enteredRubles, int enteredKopeks)

{

Rubles = enteredRubles;

Kopeks = enteredKopeks;

count++;

}

public void Show()

{

Console.WriteLine($"{Rubles} руб. {Kopeks} коп.");

}

public Money MinusKopeks(int kopeks)

{

if (kopeks>= this.Rubles\*100 + this.Kopeks)

{

Money walletMinusKopeks = new Money(0, 0);

return walletMinusKopeks;

}

else

{

Money walletMinusKopeks = new Money(0, this.Kopeks+(100\*this.Rubles) - kopeks);

return walletMinusKopeks;

}

}

public static Money operator ++ (Money walletTwo)

{

return new Money(walletTwo.Rubles, ++walletTwo.Kopeks);

}

public static Money operator --(Money walletTwo)

{

if (walletTwo.Rubles==0 & walletTwo.Kopeks == 0)

{

return new Money(0, 0);

}

else

{

return new Money(walletTwo.Rubles - 1, --walletTwo.Kopeks + 100);

}

}

public static implicit operator int(Money walletTwo)

{

return walletTwo.Rubles;

}

public static explicit operator double(Money walletTwo)

{

return walletTwo.Kopeks;

}

public static Money operator - (Money walletTwo,int kopeksMinus)

{

if (walletTwo.Rubles\*100 + walletTwo.Kopeks <= kopeksMinus)

{

return new Money(0, 0);

}

else

{

return new Money(0, walletTwo.Kopeks + (100 \* walletTwo.Rubles) - kopeksMinus);

}

}

public static bool operator ==(Money walletOne, Money walletTwo)

{

return (walletOne.Rubles\*100 +walletOne.Kopeks ==walletTwo.Rubles\*100 +walletTwo.Kopeks);

}

public static bool operator !=(Money walletOne, Money walletTwo)

{

return !(walletOne.Rubles \* 100 + walletOne.Kopeks == walletTwo.Rubles \* 100 + walletTwo.Kopeks);

}

public static Money operator -(int kopeksMinus, Money walletTwo)

{

if (walletTwo.Rubles \* 100 + walletTwo.Kopeks >= kopeksMinus)

{

return new Money(0, 0);

}

else

{

return new Money(0, kopeksMinus - (walletTwo.Rubles\*100 +walletTwo.Kopeks));

}

}

public static Money operator -(Money walletTwo, Money walletMinus)

{

Money temp = new Money(0,0);

temp.Rubles = 0;

temp.Kopeks = walletTwo.Kopeks + (100 \* walletTwo.Rubles) - (walletMinus.kopeks+walletMinus.Rubles\*100);

return temp;

}

}

}

# ПРИЛОЖЕНИЕ В **Программа MoneyArray.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics.Metrics;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace Labo9

{

public class MoneyArray

{

public static int count = 0;

Money[] arr = new Money[0];

static Random rnd = new Random();

public int Length

{

get => arr.Length;

}

public MoneyArray(int length)

{

arr = new Money[length];

for (int i = 0; i < length; i++)

{

Money arrayElement = new Money(rnd.Next(1000), rnd.Next(100));

arr[i] = arrayElement;

count++;

}

}

public MoneyArray(double length)

{

{

int lengthInt = (int)length;

arr = new Money[lengthInt];

for (int i = 0; i < arr.Length; i++)

{

Console.WriteLine($"Введите количество рублей для {i + 1}-го объекта:");

int rublesTwo = EnterNumber();

Console.WriteLine($"Введите количество копеек для {i + 1}-го объекта:");

int kopeksTwo = EnterNumber();

Money walletTwo = new Money(rublesTwo, kopeksTwo);

arr[i] = walletTwo;

count++;

}

}

}

static int EnterNumber() // Ввод целого числа

{

bool ok;

int Num;

do

{

ok = int.TryParse(Console.ReadLine(), out Num);

if (!ok)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Вы ввели неверное значение!");

Console.ForegroundColor = ConsoleColor.White;

}

} while (!ok);

if (Num <= 0)

{

return 0;

}

else

{

return Num;

}

}

public void Show()

{

for (int i = 0; i < this.Length; i++)

{

Console.Write($"({i+1}) - ");

arr[i].Show();

}

}

public Money this[int index]

{

get

{

if(index >= 0 && index < arr.Length)

{

return arr[index];

}

else

{

throw new IndexOutOfRangeException();

}

}

set

{

if (index>= 0 && index < arr.Length)

{

arr[index] = value;

}

}

}

public static int position = 0;

public static int FindMaxPosition(MoneyArray moneyArray)

{

int position = 0;

int max = 0;

for (int i =0; i < moneyArray.Length; i++)

{

if (moneyArray[i].Kopeks + moneyArray[i].Rubles \* 100 > max)

{

max = moneyArray[i].Kopeks + moneyArray[i].Rubles \* 100;

position = i;

}

}

return position;

}

}

}