Модул 7. Разработка на софтуер

Тема 1. Увод в разработката на софтуер. Преглед на трислойния модел

Задача 1.0: Реализиране на MVC приложение

В рамките на това приложение ще създадем малко примерно конзолно MVC приложение за изчисление на сметката в ресторант.

Структура

Създайте проект за конзолно приложение **ConsoleMVC**. В него добавете следните директории:

- **Program.cs** – Главна програма, нищо по-различно от обичайното ☺
- **Controllers** – Папка, в която ще се съхраняват контролер класове
  - Създайте **TipCalculatorController.cs**
- **Model** – Папка, в която ще се съхраняват моделите
  - Създайте **Tip.cs**
- **Views** – Папка, в която ще се съхраняват изгледите
  - Създайте **Display.cs**

Модел

Тук ще опишем данните, с които ще боравим в рамките на това приложение. Това са цената на поръчката и процента бакшиш, който желаем да оставим. Ще създадем полета, свойства и конструктор, които да обслужват тези данни, а също така и два публични метода – CalculateTip() и CalculateTotal(), които изчисляват съответно стойността на бакшиша и общата сума за плащане, заедно с бакшиша.

Полетата и свойствата тук са по-скоро стандартни. Единствената особеност е свойството за процент – то се грижи да превърне въведената информация в процент от 0 до 1, в случай че е въведено по-голямо число.

Следват конструкторите. Тук правим и празен конструктор, който извиква верижно другия със стойности 0 за двата параметъра.

Накрая реализираме двата метода за изчисления, които са по-скоро тривиални.

Изглед

В рамките на този клас трябва да създадем свойства за процент, стойност на поръчката, бакшиш и обща сума за плащане с включения бакшиш. Процентът и стойността ще получим при въвеждането им от потребителя през конзолата, а другите две стойности се изчисляват от модела. В рамките на класа ние ще създадем два метода GetValues() и ShowTipAndTotal().

Сега трябва да реализираме конструктора. Ще използваме този конструктор, за да дадем стойности по подразбиране на всичките свойства, но и за да извикаме метод GetValues(), чиято цел е да въведе стойности от конзолата за процента и сумата.

Сега нека минем към реализацията на GetValues(). Ще използваме обикновени входно/изходни операции.

Реализацията на ShowTipAndTotal() не е нищо по-сложно. Тук използваме {0:C}, защото си имаме работа с пари в определена валута.

Контролер

Остана да съберем отделните парченца заедно – това се случва в контролера. Първата ни задача е да направим полета от типове съответстващи на моделите и изгледите, които желаем да ползваме. Тук ще използваме Tip и Display. След това в конструктора ще създадем и обекти от въпросните класове. Важно уточнение е, че тук при създаването на обекта от Tip вече ще разполагаме с въведени от потребителя данни, тъй като извикването на конструктора Display води до въвеждането на информацията. В този смисъл тази информация може да бъде подадена на Tip като параметри.

Тук е и мястото, където ще зададем стойности на TipAmount и Total полетата от Display, извиквайки методи от модела. По този начин в контролера извършваме връзка между двата класа и предаваме данни.

Накрая извикваме метод ShowTipAndTotal(), за да се визуализира резултат от програмата.

Главна програма

Добавете следната using директива: using ConsoleMVC.Controllers;

След това трябва да създадем обект от желания контролер клас:

```
TipCalculatorController tipCalculatorController = new TipCalculatorController();
```

Решение

```
namespace ConsoleMVC
{
    public class Tip
    {
        private double amount;

        public double Amount
        {
            get { return amount; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Must be positive!");
                }
```

```
            amount = value;
        }
    }

    private double percent;

    public double Percent
    {
        get { return percent; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentException("Must be positive!");
            }
            if (value > 1)
            {
                percent = value / 100.0;
            }
            else
            {
                percent = value;
            }
        }
    }

    public Tip(double amount, double percent)
    {
        this.Amount = amount;
        this.Percent = percent;
    }

    public Tip() : this(0,0)
    {
        ;;
    }

    public double CalculateTip()
    {
        return Amount * Percent;
    }

    public double CalculateTotal()
    {
        return Amount + CalculateTip();
    }
}

public class TipCalculatorController
{
    private Tip tip = null;

    private Display display = null;

    public TipCalculatorController()
```

```
        {
            display = new Display();

            tip = new Tip(display.Amount, display.Percent);

            display.TipAmount = tip.CalculateTip();
            display.Total = tip.CalculateTotal();

            display.ShowTipAndTotal();
        }
    }

    public class Display
    {
        public double Amount { get; set; }

        public double Percent { get; set; }

        public double Total { get; set; }

        public double TipAmount { get; set; }

        public Display()
        {
            Amount = 0;
            Percent = 0;
            Total = 0;
            TipAmount = 0;
            GetValues();
        }

        public void GetValues()
        {
            Console.WriteLine("Enter the amount of the meal: ");
            this.Amount = double.Parse(Console.ReadLine());

            Console.WriteLine("Enter the percent you want to tip: ");
            this.Percent = double.Parse(Console.ReadLine());
        }

        public void ShowTipAndTotal()
        {
            Console.WriteLine("You tip is: {0:C}", TipAmount);
            Console.WriteLine("The total will be {0:C}", Total);
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            TipCalculatorController controller = new TipCalculatorController();
        }
    }
}
```

Задача 1.1. Цена за транспорт

Студент трябва да пропътува **n** километра. Той има избор измежду три вида транспорт:

- Такси. Начална такса: 0.70 лв. Дневна тарифа: 0.79 лв. / км. Нощна тарифа: 0.90 лв. / км.
- Автобус. Дневна / нощна тарифа: 0.09 лв. / км. Може да се използва за разстояния минимум 20 км.
- Влак. Дневна / нощна тарифа: 0.06 лв. / км. Може да се използва за разстояния минимум 100 км.

Напишете програма, която въвежда броя километри **n** и период от деня (ден или нощ) и изчислява цената на най-евтиния транспорт.

Вход

От конзолата се четат два реда:

- Първият ред съдържа числото **n** – брой километри – цяло число в интервала [1...5000].
- Вторият ред съдържа дума "**day**" или "**night**" – пътуване през деня или през нощта.

Изход

Да се отпечата на конзолата най-ниската цена за посочения брой километри.

Примерен вход и изход

Вход	Изход	Обяснения
5 day	4.65	Разстоянието е под 20 км → може да се ползва само такси. Началната такса е 0.70 лв. Понеже е през деня, тарифата е 0.79 лв. / км. С такси цената е: 0.70 + 5 * 0.79 = 4.65 лв.
7 night	7	Разстоянието е под 20 км → може да се ползва само такси. Началната такса е 0.70 лв. Понеже е през нощта, тарифата е 0.90 лв. / км. С такси цената е: 0.70 + 7 * 0.90 = 7.00 лв.
25 day	2.25	Разстоянието е над 20 км → може да се ползва автобус, но не може да се ползва влак. Автобусът е най-евтиния възможен вариант. С автобус цената е: 25 * 0.09 = 2.25 лв.
180 night	10.8	Разстоянието е над 100 км → може да се ползва влак. Влакът е най-евтиният възможен вариант за пътуване. С влак цената е: 180 * 0.06 = 10.80 лв.

Решение

```
namespace Transport
{
    public class Model
    {
        private int kilometers;

        public int Kilometers
        {
            get { return kilometers; }
```

```
        set { kilometers = value; }
    }

    private string time;

    public string Time
    {
        get { return time; }
        set { time = value; }
    }

    private double startPrice;

    public double StartPrice
    {
        get { return startPrice; }
        set { startPrice = value; }
    }

    public Model(int kilometres, string time)
    {
        Kilometers = kilometres;
        Time = time;
    }

    public Model() : this(0, "")
    {
        // nope
    }

    public double CalculateCheapest()
    {
        double pricePerKm = 0;
        if (kilometers < 20)
        {
            // Такси
            startPrice = 0.70;
            if (time == "day")
                pricePerKm = 0.79;
            else
                pricePerKm = 0.90;
        }
        else if (kilometers >= 20 && kilometers <= 100)
        {
            // Автобус
            startPrice = 0;
            pricePerKm = 0.09;
        }
        else
        {
            // Влак
            startPrice = 0;
            pricePerKm = 0.06;
        }
        return pricePerKm;
```

```
        }

        public double CalculatePrice()
        {
            return CalculateCheapest() * kilometers + startPrice;
        }
    }

    public class Controller
    {
        private View view;
        private Model model;

        public Controller()
        {
            view = new View();
            model = new Model(view.kilometers, view.time);

            view.totalPrice = model.CalculatePrice();

            view.ShowCheapestWayToTravel();
        }
    }

    public class View
    {
        public string time { get; set; }

        public int kilometers { get; set; }

        public double totalPrice { get; set; }

        public View()
        {
            time = "";
            kilometers = 0;
            totalPrice = 0;
            GetValues();
        }

        public void GetValues()
        {
            Console.WriteLine("Enter the kilometers you will travel: ");
            kilometers = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter the phase of the day you will travel
(day/night)");
            time = Console.ReadLine();
        }

        public void ShowCheapestWayToTravel()
        {
            Console.WriteLine($"Total price = {totalPrice:f2}");
        }
    }
```

```
    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.2. Навреме за изпит

Студент трябва да отиде на изпит в определен час (например в 9:30 часа). Той идва в изпитната зала в даден час на пристигане (например 9:40). Счита се, че студентът е дошъл навреме, ако е пристигнал в часа на изпита или до половин час преди това. Ако е пристигнал по-рано повече от 30 минути, той е подранил. Ако е дошъл след часа на изпита, той е закъснял. Напишете програма, която въвежда време на изпит и време на пристигане и отпечатва дали студентът е дошъл навреме, дали е подранил или е закъснял и с колко часа или минути е подранил или закъснял.

Вход

От конзолата се четат 4 цели числа (по едно на ред):

- Първият ред съдържа час на изпита – цяло число от 0 до 23.
- Вторият ред съдържа минута на изпита – цяло число от 0 до 59.
- Третият ред съдържа час на пристигане – цяло число от 0 до 23.
- Четвъртият ред съдържа минута на пристигане – цяло число от 0 до 59.

Изход

На първият ред отпечатайте:

- **"Late", ако студентът пристига по-късно от часа на изпита.**
- "On time", ако студентът пристига точно в часа на изпита или до 30 минути по-рано.
- "Early", ако студентът пристига повече от 30 минути преди часа на изпита.

Ако студентът пристига с поне минута разлика от часа на изпита, отпечатайте на следващия ред:

- "`mm` minutes before the start" за идване по-рано с по-малко от час.
- "`hh:mm` hours before the start" за подраняване с 1 час или повече. Минутите винаги печатайте с 2 цифри, например "1:05".
- "`mm` minutes after the start" за закъснение под час.
- "`hh:mm` hours after the start" за закъснение от 1 час или повече. Минутите винаги печатайте с 2 цифри, например "1:03".

Примерен вход и изход

Вход	Изход
9 30 9	Late 20 minutes after the start

Вход	Изход
9 00 10	Late 1:30 hours after the start

Вход	Изход
10 00 10	On time

50	
9 00 8 30	On time 30 minutes before the start
16 00 15 00	Early 1:00 hours before the start

30	
14 00 13 55	On time 5 minutes before the start
11 30 8 12	Early 3:18 hours before the start

00	
11 30 10 55	Early 35 minutes before the start
11 30 12 29	Late 59 minutes after the start

Решение

```
namespace OnTimeForExam
{
    public class Model
    {
        private DateTime startTime;
        private DateTime arrivalTime;

        public int startHour;

        public int StartHour
        {
            get { return startHour; }
            set
            {
                if (value >= 0 && value <= 23)
                {
                    startHour = value;
                }
                else
                {
                    throw new Exception("Incorrect Start Hour Input");
                }
            }
        }

        private int startMinutes;

        public int StartMinutes
        {
            get { return startMinutes; }
            set
            {
                if (value >= 0 && value <= 59)
                {
                    startMinutes = value;
                }
                else
                {
                    throw new Exception("Incorrect Start Minutes Input");
                }
            }
        }
```

```
        private int arrivalHour;
        public int ArrivalHour
        {
            get { return arrivalHour; }
            set
            {
                if (value >= 0 && value <= 23)
                {
                    arrivalHour = value;
                }
                else
                {
                    throw new Exception("Incorrect Arrival Hour Input");
                }
            }
        }

        private int arrivalMinutes;
        public int ArrivalMinutes
        {
            get { return arrivalMinutes; }
            set
            {
                if (value >= 0 && value <= 59)
                {
                    arrivalMinutes = value;
                }
                else
                {
                    throw new Exception("Incorrect Arrival Minutes Input");
                }
            }
        }

        public Model(int sh, int sm, int ah, int am)
        {
            StartHour = sh;
            StartMinutes = sm;
            ArrivalHour = ah;
            ArrivalMinutes = am;

            startTime = new DateTime(DateTime.Now.Year, DateTime.Now.Month,
DateTime.Now.Month, sh, sm, 0);
            arrivalTime = new DateTime(DateTime.Now.Year, DateTime.Now.Month,
DateTime.Now.Month, ah, am, 0);
        }

        public string Calculate()
        {
            string result = String.Empty;
            if (DateTime.Compare(arrivalTime, startTime) <= 0)
            {
                int diffHours = (startTime - arrivalTime).Hours;
                int diffMinuts = (startTime - arrivalTime).Minutes;
                if (diffMinuts <= 30 && diffHours == 0)
```

```
                {
                    return "On time\n";
                }
                else
                {
                    result += "Early\n";
                }
                var append = String.Empty;
                if (diffMinuts < 10) append = "0";
                if (diffMinuts > 0 && diffHours == 0)
                {
                    result += $"{append}{diffMinuts} minutes before the start";
                }
                else result += $"{diffHours}:{append}{diffMinuts} hours before the
start";
            }
            else
            {
                result += "Late\n";
                int diffHours = (startTime - arrivalTime).Hours;
                int diffMinuts = (startTime - arrivalTime).Minutes;

                var append = String.Empty;
                if (diffMinuts < 10) append = "0";
                if (diffMinuts > 0 && diffHours == 0)
                {
                    result += $"{append}{diffMinuts} minutes after  the start";
                }
                else result += $"{diffHours}:{append}{diffMinuts} hours after  the
start";
            }
            return result;
        }
    }

    public class View
    {
        public int StartHour { get; set; }
        public int StartMinutes { get; set; }
        public int ArrivalHour { get; set; }
        public int ArrivalMinutes { get; set; }
        public string Response { get; set; }

        public View()
        {
            StartHour = 0;
            StartMinutes = 0;
            ArrivalHour = 0;
            ArrivalMinutes = 0;
            Input();
        }

        public void Input()
        {
            Console.Write("Enter the exam hour start: ");
```

```
            StartHour = int.Parse(Console.ReadLine());

            Console.Write("Enter the exam minutes start: ");
            StartMinutes = int.Parse(Console.ReadLine());

            Console.Write("Enter arrival hour: ");
            ArrivalHour = int.Parse(Console.ReadLine());

            Console.Write("Enter arrival minutes: ");
            ArrivalMinutes = int.Parse(Console.ReadLine());
        }

        public void ShowResponse()
        {
            Console.WriteLine(Response);
        }
    }

    public class Controller
    {
        private View view;
        private Model model;

        public Controller()
        {
            view = new View();
            while (model is null)
            {
                try
                {
                    model = new Model
                    (
                        view.StartHour,
                        view.StartMinutes,
                        view.ArrivalHour,
                        view.ArrivalMinutes
                    );
                }
                catch (Exception e)
                {
                    view.Response = e.Message;
                    view.ShowResponse();
                }
            }
            view.Response = model.Calculate();
            view.ShowResponse();
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
```

```
    }
}
```

Задача 1.3. Хистограма

Дадени са `n` цели числа в интервала [1...1000]. От тях някакъв процент `p1` са под 200, друг процент `p2` са от 200 до 399, друг процент `p3` са от 400 до 599, друг процент `p4` са от 600 до 799 и останалите `p5` процента са от 800 нагоре. Да се напише програма, която изчислява и отпечатва процентите `p1`, `p2`, `p3`, `p4` и `p5`.

Пример: имаме n = 20 числа: 53, 7, 56, 180, 450, 920, 12, 7, 150, 250, 680, 2, 600, 200, 800, 799, 199, 46, 128, 65. Получаваме следното разпределение и визуализация:

Диапазон	Числа в диапазона	Брой числа	Процент
< 200	53, 7, 56, 180, 12, 7, 150, 2, 199, 46, 128, 65	12	p1 = 12 / 20 * 100 = 60.00%
200 … 399	250, 200	2	p2 = 2 / 20 * 100 = 10.00%
400 … 599	450	1	p3 = 1 / 20 * 100 = 5.00%
600 … 799	680, 600, 799	3	p4 = 3 / 20 * 100 = 15.00%
≥ 800	920, 800	2	p5 = 2 / 20 * 100 = 10.00%

Вход

На първия ред от входа стои цялото число **n** (1 ≤ **n** ≤ 1000) – брой числа. На следващите **n** реда стои по едно цяло число в интервала [1...1000] – числата върху които да бъде изчислена хистограмата.

Изход

Да се отпечата на конзолата хистограмата – 5 реда, всеки от които съдържа число между 0% и 100%, с точност две цифри след десетичната точка, например 25.00%, 66.67%, 57.14%.

Примерен вход и изход

Вход	Изход
3 1 2 999	66.67% 0.00% 0.00% 0.00% 33.33%

Вход	Изход
4 53 7 56 999	75.00% 0.00% 0.00% 0.00% 25.00%

Вход	Изход
7 800 801 250 199 399 599 799	14.29% 28.57% 14.29% 14.29% 28.57%

Вход	Изход
9 367 99 200 799 999 333 555 111 9	33.33% 33.33% 11.11% 11.11% 11.11%

Вход	Изход
14 53 7 56 180 450 920 12 7 150 250 680 2 600 200	57.14% 14.29% 7.14% 14.29% 7.14%

Решение

```
namespace Histogram
{
    public class Model
    {
        private double count;

        public double Count
        {
            get { return count; }
            set { count = value; }
        }

        private List<int> numbers;

        public List<int> Numbers
        {
            get { return numbers; }
            set { numbers = value; }
        }

        public Model(double count, List<int> numbers)
        {
            this.count = count;
            this.numbers = numbers;
        }

        public Model() : this(0, new List<int>())
        {
            // nope
        }

        public string FindPercentages()
        {
            List<int> p1 = numbers.Where(x => x < 200).ToList();
            List<int> p2 = numbers.Where(x => x < 400 && x >= 200).ToList();
            List<int> p3 = numbers.Where(x => x < 600 && x >= 400).ToList();
            List<int> p4 = numbers.Where(x => x < 800 && x >= 600).ToList();
            List<int> p5 = numbers.Where(x => x >= 800).ToList();

            return $"{(p1.Count / this.count) * 100:f2}%" + Environment.NewLine +
                   $"{(p2.Count / this.count) * 100:f2}%" + Environment.NewLine +
                   $"{(p3.Count / this.count) * 100:f2}%" + Environment.NewLine +
                   $"{(p4.Count / this.count) * 100:f2}%" + Environment.NewLine +
                   $"{(p5.Count / this.count) * 100:f2}%" + Environment.NewLine;
        }
    }

    public class View
    {
        public double count { get; set; }
        public List<int> numbers { get; set; }
        public string percents { get; set; }
```

```
        public View()
        {
            count = 0;
            numbers = new List<int>();
            percents = String.Empty;
            Input();
        }

        public void Input()
        {
            Console.WriteLine("Enter how many numbers you want: ");
            count = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter the numbers on new lines:");
            for (int i = 0; i < count; i++)
            {
                numbers.Add(int.Parse(Console.ReadLine()));
            }
        }

        public void ShowPercents()
        {
            Console.WriteLine(percents);
        }
    }

    public class Controller
    {
        private View view;
        private Model model;

        public Controller()
        {
            view = new View();
            model = new Model(view.count, view.numbers);

            view.percents = model.FindPercentages();

            view.ShowPercents();
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.4. Генератор за тъпи пароли

Да се напише програма, която въвежда две цели числа ***n*** и ***l*** и генерира по азбучен ред всички възможни "тъпи" пароли, които се състоят от следните 5 символа:

- Символ 1: цифра от **1** до ***n***.
- Символ 2: цифра от **1** до ***n***.
- Символ 3: малка буква измежду първите ***l*** букви на латинската азбука.
- Символ 4: малка буква измежду първите ***l*** букви на латинската азбука.
- Символ 5: цифра от 1 до **n**, по-голяма от първите 2 цифри.

Вход

Входът се чете от конзолата и се състои от две цели числа ***n*** и ***l*** в интервала [1...9], по едно на ред.

Изход

На конзолата трябва да се отпечатат всички "тъпи" пароли по азбучен ред, разделени с интервал.

Примерен вход и изход

вход	изход
2 4	11aa2 11ab2 11ac2 11ad2 11ba2 11bb2 11bc2 11bd2 11ca2 11cb2 11cc2 11cd2 11da2 11db2 11dc2 11dd2
3 1	11aa2 11aa3 12aa3 21aa3 22aa3
3 2	11aa2 11aa3 11ab2 11ab3 11ba2 11ba3 11bb2 11bb3 12aa3 12ab3 12ba3 12bb3 21aa3 21ab3 21ba3 21bb3 22aa3 22ab3 22ba3 22bb3
4 2	11aa2 11aa3 11aa4 11ab2 11ab3 11ab4 11ba2 11ba3 11ba4 11bb2 11bb3 11bb4 12aa3 12aa4 12ab3 12ab4 12ba3 12ba4 12bb3 12bb4 13aa4 13ab4 13ba4 13bb4 21aa3 21aa4 21ab3 21ab4 21ba3 21ba4 21bb3 21bb4 22aa3 22aa4 22ab3 22ab4 22ba3 22ba4 22bb3 22bb4 23aa4 23ab4 23ba4 23bb4 31aa4 31ab4 31ba4 31bb4 32aa4 32ab4 32ba4 32bb4 33aa4 33ab4 33ba4 33bb4

Решение

```
namespace PasswordGenerator
{
    public class Model
    {
        private int n;

        public int N
        {
            get { return n; }
            set
            {
                if (value > 0 && value < 10)
                {
                    n = value;
                }
                else
                {
                    throw new Exception("Incorrrect value");
                }
            }
        }

        private int l;
```

```
        public int L
        {
            get { return l; }
            set
            {
                if (value > 0 && value < 10)
                {
                    l = value;
                }
                else
                {
                    throw new Exception("Incorrrect value");
                }
            }
        }

        public Model(int n, int l)
        {
            N = n;
            L = l;
        }

        public string CalculatePasswords()
        {
            string result = String.Empty;

            for (int a = 1; a <= n; a++)
            {
                for (int b = 1; b <= n; b++)
                {
                    for (char c = 'a'; c < 'a' + l; c++)
                    {
                        for (char d = 'a'; d < 'a' + l; d++)
                        {
                            for (int e = 1; e <= n; e++)
                            {
                                if (e > a && e > b)
                                {
                                    result += $"{a}{b}{c}{d}{e} ";
                                }
                            }
                        }
                    }
                }
            }

            return result;
        }
    }

    public class View
    {
        public int n { get; set; }
        public int l { get; set; }
```

```
        public string Response { get; set; }

        public View()
        {
            n = 0;
            l = 0;
            Response = String.Empty;
        }

        public void Input()
        {
            Console.Write("n = ");
            n = int.Parse(Console.ReadLine());

            Console.Write("l = ");
            l = int.Parse(Console.ReadLine());
        }

        public void ShowResponse()
        {
            Console.WriteLine(Response);
        }
    }

    public class Controller
    {
        public View view = null;
        public Model model = null;

        public Controller()
        {
            view = new View();

            view.Input();

            model = new Model(view.n, view.l);

            view.Response = model.CalculatePasswords();

            view.ShowResponse();
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.5. Зеленчукова борса

Градинар продавал реколтата от градината си на зеленчуковата борса. Продава зеленчуци за N лева на килограм и плодове за M лева за килограм.

Напишете програма, която да пресмята приходите от реколтата в евро (ако приемем, че едно евро е равно на 1.94лв).

Вход

От конзолата се четат 4 числа, по едно на ред:

- Първи ред – Цена за килограм зеленчуци – число с плаваща запетая
- Втори ред – Цена за килограм плодове – число с плаваща запетая
- Трети ред – Общо килограми на зеленчуците – цяло число
- Четвърти ред – Общо килограми на плодовете – цяло число

Ограничения: Всички числа ще са в интервала от 0.00 до 1000.00

Изход

Да се отпечата на конзолата едно число с плаваща запетая: приходите от всички плодове и зеленчуци в евро.

Примерен вход и изход

Вход	Изход	Обяснения
0.194 19.4 10 10	101	Зелечуците струват – 0.194лв. * 10кг. = 1.94лв. Плодовете струват – 19.4лв. * 10кг. = 194лв. Общо – 195.94лв. = 101евро
1.5 2.5 10 10	20.6185567010309	

Решение

```
namespace GroceryStore
{
    public class Model
    {
        private double priceOfVegetables;

        public double PriceOfVegetables
        {
            get { return priceOfVegetables; }
            set
            {
                if (value < 0)
                {
                    throw new Exception("Must be positive numver");
                }
                priceOfVegetables = value;
            }
        }

        private double priceOfFruit;

        public double PriceOfFruit
        {
```

```
            get { return priceOfFruit; }
            set
            {
                if (value < 0)
                {
                    throw new Exception("Must be positive numver");
                }
                priceOfFruit = value;
            }
        }

        private int kilosVegetables;

        public int KilosVegetables
        {
            get { return kilosVegetables; }
            set
            {
                if (value < 0)
                {
                    throw new Exception("Must be positive numver");
                }
                kilosVegetables = value;
            }
        }

        private int kilosFruits;

        public int KilosFruits
        {
            get { return kilosFruits; }
            set
            {
                if (value < 0)
                {
                    throw new Exception("Must be positive numver");
                }
                kilosFruits = value;
            }
        }

        public Model(double priceOfVegetables, double priceOfFruit, int
kilosVegetables, int kilosFruits)
        {
            this.priceOfFruit = priceOfFruit;
            this.priceOfVegetables = priceOfVegetables;
            this.kilosVegetables = kilosVegetables;
            this.kilosFruits = kilosFruits;
        }

        public Model() : this(0, 0, 0, 0)
        {
            // nope
        }
```

```
        public double CalculateTotal()
        {
            return ((priceOfVegetables * kilosVegetables) + (priceOfFruit *
kilosFruits)) / 1.94;
        }
    }

    public class View
    {
        public double priceOfVegetables { get; set; }

        public double priceOfFruit { get; set; }

        public int kilosVegetables { get; set; }

        public int kilosFruits { get; set; }

        public double Total { get; set; }

        public View()
        {
            priceOfVegetables = 0;
            priceOfFruit = 0;
            kilosVegetables = 0;
            kilosFruits = 0;
            Total = 0;
            Input();
        }

        public void Input()
        {
            Console.WriteLine("Enter price per kilogram of the vegetables: ");
            priceOfVegetables = double.Parse(Console.ReadLine());

            Console.WriteLine("Enter price per kilogram of the fruits: ");
            priceOfFruit = double.Parse(Console.ReadLine());

            Console.WriteLine("Enter how many kilos vegetables: ");
            kilosVegetables = int.Parse(Console.ReadLine());

            Console.WriteLine("Enter how many kilos fruits: ");
            kilosFruits = int.Parse(Console.ReadLine());
        }

        public void ShowTotal()
        {
            Console.WriteLine(Total);
        }
    }

    public class Controller
    {
        public Model model { get; set; }

        public View view { get; set; }
```

```
        public Controller()
        {
            view = new View();

            model = new Model
            (
                view.priceOfVegetables,
                view.priceOfFruit,
                view.kilosVegetables,
                view.kilosFruits
            );

            view.Total = model.CalculateTotal();

            view.ShowTotal();
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.6. Тръби в басейн

Басейн с обем V има две тръби от които се пълни. Всяка тръба има определен дебит (литрите вода минаващи през една тръба за един час). Работникът пуска тръбите едновременно и излиза за N часа. Напишете програма, която изкарва състоянието на басейна, в момента, когато работникът се върне.

Вход

От конзолата се четат четири реда:

- Първият ред съдържа числото **V** – Обем на басейна в литри – цяло число в интервала [1...10000].
- Вторият ред съдържа числото P1 – дебит на първата тръба за час – цяло число в интервала [1...5000].
- Третият ред съдържа числото P2 – дебит на втората тръба за час– цяло число в интервала [1...5000].
- Четвъртият ред съдържа числото H – часовете които работникът отсъства – число с плаваща запетая в интервала [1.0...24.00]

Изход

Да се отпечата на конзолата едно от двете възможни състояния:

- До колко се е запълнил басейна и коя тръба с колко процента е допринесла. Всички проценти се свеждат до цяло число (без закръгляне).

- "The pool is [x]% full. Pipe 1: [y]%. Pipe 2: [z]%."
- Ако басейнът се е препълнил – с колко литра е прелял за даденото време, число с плаваща запетая
  - "For [x] hours the pool overflows with [y] liters."

* Имайте предвид, че поради свеждането до цяло число се губят данни и нормално сборът на процентите да е 99%, а не 100%.

Примерен вход и изход

Вход	Изход	Обяснения
1000 100 120 3	The pool is 66% full. Pipe 1: 45%. Pipe 2: 54%.	За 3 часа: Първата тръба е напълнила – 300 л. Втората тръба е напълнила – 360 л. Общо – 660 л. < 1000 л. => 66% са запълнени Първата тръба е допринесла с 45% (300 от 660 л.). Втората тръба е допринесла с 54% (360 от 660 л.).
100 100 100 2.5	For 2.5 hours the pool overflows with 400 liters.	За 2.5 часа: Първата тръба е напълнила – 250 л. Втората тръба е напълнила – 250 л. Общо – 500 л. > 100 л. => 400 л. са преляли.

Решение

```
namespace Pool
{
    public class Model
    {
        private int v;

        public int V
        {
            get { return v; }
            set
            {
                if (value >= 1 && value <= 10000)
                {
                    v = value;
                }
                else
                {
                    throw new Exception("Incorrrect value");
                }
            }
        }

        private int p1;

        public int P1
        {
            get { return p1; }
            set
```

```
        {
            if (value >= 1 && value <= 5000)
            {
                p1 = value;
            }
            else
            {
                throw new Exception("Incorrrect value");
            }
        }
    }

    private int p2;

    public int P2
    {
        get { return p2; }
        set
        {
            if (value >= 1 && value <= 5000)
            {
                p2 = value;
            }
            else
            {
                throw new Exception("Incorrrect value");
            }
        }
    }

    public double h;
    public double H
    {
        get { return h; }
        set
        {
            if (value >= 1 && value <= 24)
            {
                h = value;
            }
            else
            {
                throw new Exception("Incorrrect h value");
            }
        }
    }

    public Model(int v, int p1, int p2, double h)
    {
        V = v;
        P1 = p1;
        P2 = p2;
        H = h;
    }
```

```
        public string CalculateResponse()
        {
            string result = String.Empty;
            double pipe1 = p1 * h;
            double pipe2 = p2 * h;

            var filled = pipe1 + pipe2;

            if (filled <= v)
            {
                result = $"The pool is {(int)(filled / v * 100)} % full.Pipe 1:
{(int)(pipe1 / filled * 100)}%. Pipe 2: {(int)(pipe2 / filled * 100)}%.";
            }
            else
            {
                result = $"For {h} hours the pool overflows with {filled - v}
liters.";
            }

            return result;
        }
    }

    public class View
    {
        public int v { get; set; }

        public int p1 { get; set; }

        public int p2 { get; set; }

        public double h { get; set; }

        public string Response { get; set; }

        public View()
        {
            v = 0;
            p1 = 0;
            p2 = 0;
            h = 0;
            Input();
        }

        private void Input()
        {
            Console.Write("V = ");
            v = int.Parse(Console.ReadLine());

            Console.Write("P1 = ");
            p1 = int.Parse(Console.ReadLine());

            Console.Write("P2 = ");
            p2 = int.Parse(Console.ReadLine());
```

```
            Console.Write("H = ");
            h = double.Parse(Console.ReadLine());
        }

        public void ShowResponse()
        {
            Console.WriteLine(Response);
        }
    }

    public class Controller
    {
        public Model model { get; set; }

        public View view { get; set; }

        public Controller()
        {
            view = new View();

            model = new Model
            (
                view.v,
                view.p1,
                view.p2,
                view.h
            );

            view.Response = model.CalculateResponse();

            view.ShowResponse();
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.7. Пътешествие

Странно, но повечето хора си плануват от рано почивката. Млад програмист разполага с определен бюджет и свободно време в даден сезон. Напишете програма, която да приема на входа бюджета и сезона, а на изхода да изкарва, къде ще почива програмиста и колко ще похарчи.
Бюджета определя дестинацията, а сезона определя колко от бюджета ще изхарчи. Ако е лято ще почива на къмпинг, а зимата в хотел. Ако е в Европа, независимо от сезона ще почива в хотел. Всеки къмпинг или хотел, според дестинацията, има собствена цена която отговаря на даден процент от бюджета:

- При 100лв. или по-малко – някъде в България
  - Лято – 30% от бюджета
  - Зима – 70% от бюджета
- При 1000лв. или по малко – някъде на Балканите
  - Лято – 40% от бюджета
  - Зима – 80% от бюджета
- При повече от 1000лв. – някъде из Европа
  - При пътуване из Европа, независимо от сезона ще похарчи 90% от бюджета.

Вход

Входът се чете от конзолата и се състои от два реда:

- Първи ред – Бюджет, реално число в интервала [10.00...5000.00].
- Втори ред – Един от двата възможни сезона: „summer" или "winter"

Изход

На конзолата трябва да се отпечатат два реда.

- Първи ред – „Somewhere in [дестинация]" измежду "Bulgaria","Balkans" и "Europe"
- Втори ред – "{Вид почивка} – {Похарчена сума}"
  - Почивката може да е между „Camp" и „Hotel"
  - Сумата трябва да е закръглена с точност до вторият знак след запетаята.

Примерен вход и изход

вход	изход
50 summer	Somewhere in Bulgaria Camp - 15.00
75 winter	Somewhere in Bulgaria Hotel - 52.50
312 summer	Somewhere in Balkans Camp - 124.80
678.53 winter	Somewhere in Balkans Hotel - 542.82
1500 summer	Somewhere in Europe Hotel - 1350.00

Решение

```
namespace Trip
{
    public class Model
    {
        private double budget;

        public double Budget
        {
            get { return budget; }
            set { budget = value; }
```

```
}

private string season;

public string Season
{
    get { return season; }
    set { season = value; }
}

private string place;

public string Place
{
    get { return place; }
    set { place = value; }
}

private double expenses;

public double Expenses
{
    get { return expenses; }
    set { expenses = value; }
}

private string facility;

public string Facility
{
    get { return facility; }
    set { facility = value; }
}

public Model(double budget, string season)
{
    this.budget = budget;
    this.season = season;
    this.place = "";
    this.facility = "";
    this.expenses = 0.0;
}

public Model() : this(0, "")
{
    // Empty
}

public string DefinePlace()
{
    if (budget < 100)
    {
        place = "Bulgaria";
    }
    else if (budget >= 100 && budget < 1000)
```

```
            {
                place = "Balkans";
            }
            else
            {
                place = "Europe";
            }
            return place;
        }

        public double CalculateExpenses()
        {
            switch (this.place)
            {
                case "Bulgaria":
                    if (season == "summer")
                    {
                        this.facility = "Camp";
                        this.expenses = budget * 0.3;
                    }
                    else
                    {
                        this.facility = "Hotel";
                        this.expenses = budget * 0.7;
                    }
                    break;

                case "Balkans":
                    if (season == "summer")
                    {
                        this.facility = "Camp";
                        this.expenses = budget * 0.4;
                    }
                    else
                    {
                        this.facility = "Hotel";
                        this.expenses = budget * 0.8;
                    }
                    break;

                case "Europe":
                    this.facility = "Hotel";
                    this.expenses = budget * 0.9;
                    break;
            }
            return expenses;
        }
    }

    public class View
    {
        public double budget { get; set; }

        public string season { get; set; }
```

```
        public string place { get; set; }

        public double expenses { get; set; }

        public string facility { get; set; }

        public View()
        {
            this.budget = 0;
            this.season = "";
            this.place = "";
            this.facility = "";
            this.expenses = 0.0;
            GetValues();
        }

        public void GetValues()
        {
            Console.WriteLine("Enter budget: ");
            this.budget = double.Parse(Console.ReadLine());

            Console.WriteLine("Enter season (summer/winter)");
            this.season = Console.ReadLine();
        }

        public void ShowResult()
        {
            Console.WriteLine($"Somewhere in {place}");
            Console.WriteLine($"{facility} - {expenses:f2}");
        }
    }

    public class Controller
    {
        public Model model { get; set; }

        public View view { get; set; }

        public Controller()
        {
            view = new View();

            model = new Model
            (
                view.budget,
                view.season
            );

            view.place = model.DefinePlace();
            view.expenses = model.CalculateExpenses();
            view.facility = model.Facility;

            view.ShowResult();
        }
    }
```

```
    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.8. Деление без остатък

Дадени са `n` цели числа в интервала [1...1000]. От тях някакъв процент `p1` се делят без остатък на 2, друг процент `p2` се делят без остатък на 3, друг процент `p3` се делят без остатък на 4. Да се напише програма, която изчислява и отпечатва процентите `p1`, `p2` и `p3`.

Пример: имаме n = 10 числа: 680, 2, 600, 200, 800, 799, 199, 46, 128, 65. Получаваме следното разпределение и визуализация:

Деление без остатък на:	Числа в диапазона	Брой числа	Процент
2	`680, 2, 600, 200, 800, 46, 128`	7	`p1 = 7.0 / 10 * 100 = 70.00%`
3	`600`	1	`p2 = 1 / 10 * 100 = 10.00%`
4	`680, 600, 200, 800, 128`	5	`p3 = 5 / 10 * 100 = 50.00%`

Вход

На първия ред от входа стои цялото число **n** (1 ≤ **n** ≤ 1000) – брой числа. На следващите **n** реда стои по едно цяло число в интервала [1...1000] – числата които да бъдат проверени на колко се делят.

Изход

Да се отпечатат на конзолата 3 реда, всеки от които съдържа процент между 0% и 100%, с точност две цифри след десетичната точка, например 25.00%, 66.67%, 57.14%.

- На първият ред – процентът на числата които се делят на 2
- На вторият ред – процентът на числата които се делят на 3
- На третият ред – процентът на числата които се делят на 4

Примерен вход и изход

Вход	Изход
10 680 2 600 200	70.00% 10.00% 50.00%

Вход	Изход
3 3 6 9	33.33% 100.00% 0.00%

800 799 199 46 128 65				

Решение

```
namespace Div
{
    public class Model
    {
        private List<int> numbers;
        public List<int> Numbers
        {
            get { return numbers; }
            set
            {
                if (value.Count(x => x < 1 && x > 1000) == 0)
                {
                    numbers = value;
                }
                else
                {
                    throw new Exception("Incorrrect number value");
                }
            }
        }

        private double p1, p2, p3, n;

        public Model(List<int> numbers)
        {
            Numbers = numbers;
            p1 = 0;
            p2 = 0;
            p3 = 0;
            n = numbers.Count;
        }

        public Model() : this(new List<int>())
        {
            // Empty
        }

        public string CalculateResponse()
        {
            p1 = (double)numbers.Count(x => x % 2 == 0) / n * 100;
            p2 = (double)numbers.Count(x => x % 3 == 0) / n * 100;
            p3 = (double)numbers.Count(x => x % 4 == 0) / n * 100;
            return $"{Math.Round(p1, 2):f2}%" + Environment.NewLine +
                   $"{Math.Round(p2, 2):f2}%" + Environment.NewLine +
                   $"{Math.Round(p3, 2):f2}%\n";
```

```
    }
}

public class View
{
    public List<int> numbers { get; set; }

    public string Response { get; set; }

    public View()
    {
        numbers = new List<int>();
        Response = string.Empty;
        Input();
    }

    public void Input()
    {
        int n = numbers.Count;
        while (n < 1 || n > 1000)
        {
            Console.Write("n = ");
            n = int.Parse(Console.ReadLine());
        }
        for (int i = 0; i < n; i++)
        {
            Console.Write($"n[{i + 1}] = ");
            var num = int.Parse(Console.ReadLine());
            numbers.Add(num);
        }
    }

    public void ShowResponse()
    {
        Console.WriteLine(Response);
    }
}

public class Controller
{
    public View view { get; set; }

    public Model model { get; set; }

    public Controller()
    {
        view = new View();

        model = new Model
        (
            view.numbers
        );

        view.Response = model.CalculateResponse();
```

```
            view.ShowResponse();
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.9. Магически числа

Да се напише програма, която въвежда едно цяло „магическо“ число и изкарва всички възможни 6-цифрени числа, за които произведението на неговите цифри е равно на „магическото“ число.

Пример: „Магическо число“ -> 2

- 111112 -> 1 * 1 * 1 * 1 * 1 * 2 = 2
- 111121 -> 1 * 1 * 1 * 1 * 2 * 1 = 2
- 111211 -> 1 * 1 * 1 * 2 * 1 * 1 = 2
- 112111 -> 1 * 1 * 2 * 1 * 1 * 1 = 2
- 121111 -> 1 * 2 * 1 * 1 * 1 * 1 = 2
- 211111 -> 2 * 1 * 1 * 1 * 1 * 1 = 2

Вход

Входът се чете от конзолата и се състои от едно цяло число в интервала [1...600000].

Изход

На конзолата трябва да се отпечатат всички “магически” числа, разделени с интервал.

Примерен вход и изход

вход	изход
2	111112 111121 111211 112111 121111 211111
8	111118 111124 111142 111181 111214 111222 111241 111412 111421 111811 112114 112122 112141 112212 112221 112411 114112 114121 114211 118111 121114 121122 121141 121212 121221 121411 122112 122121 122211 124111 141112 141121 141211 142111 181111 211114 211122 211141 211212 211221 211411 212112 212121 212211 214111 221112 221121 221211 222111 241111 411112 411121 411211 412111 421111 811111
531441	999999

Решение

```
namespace MagicNumbers
{
    public class Model
    {
        private int magicNumber;
```

```
public int MagicNumber
{
    get { return magicNumber; }
    set { magicNumber = value; }
}

private List<string> magicNumbers;

public List<string> MagicNumbers
{
    get { return magicNumbers; }
    set { magicNumbers = value; }
}


public Model(int magicNumber)
{
    this.magicNumber = magicNumber;
    this.magicNumbers = new List<string>();
}

public Model() : this(0)
{
    // Empty
}

public List<string> GenerateCombinations()
{
    string comb = "";
    for (int i1 = 0; i1 < 10; i1++)
    {
        for (int i2 = 0; i2 < 10; i2++)
        {
            for (int i3 = 0; i3 < 10; i3++)
            {
                for (int i4 = 0; i4 < 10; i4++)
                {
                    for (int i5 = 0; i5 < 10; i5++)
                    {
                        for (int i6 = 0; i6 < 10; i6++)
                        {
                            if (i1 * i2 * i3 * i4 * i5 * i6 == magicNumber)
                            {
                                comb = $"{i1}{i2}{i3}{i4}{i5}{i6}";
                                magicNumbers.Add(comb);
                            }
                        }
                    }
                }
            }
        }
    }
    return magicNumbers;
}
```

```
    }

    public class View
    {
        public int magicNumber { get; set; }

        public List<string> magicNumbers { get; set; }

        public View()
        {
            magicNumber = 0;
            magicNumbers = new List<string>();
            GetValues();
        }

        public void GetValues()
        {
            Console.WriteLine("Enter the magic number:");
            this.magicNumber = int.Parse(Console.ReadLine());
        }

        public void ShowResults()
        {
            Console.WriteLine(string.Join(" ", magicNumbers));
        }
    }

    public class Controller
    {
        public View view { get; set; }

        public Model model { get; set; }

        public Controller()
        {
            view = new View();

            model = new Model( view.magicNumber );

            view.magicNumbers = model.GenerateCombinations();

            view.ShowResults();
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Задача 1.10. Ремонт на плочки

На площадката пред жилищен блок трябва да се поставят плочки. Площадката е с форма на квадрат със страна N метра. Плочките са широки „W“ метра и дълги „L“ метра. На площадката има една пейка с ширина M метра и дължина O метра. Под нея не е нужно да се слагат плочки. Всяка плочка се поставя за 0.2 минути.

Напишете програма, която чете от конзолата размерите на площадката, плочките и пейката и пресмята колко плочки са необходими да се покрие площадката и пресмята времето за поставяне на всички плочки.

Пример: площадка с размер 20м. има площ 400кв.м. Пейка широка 1м. и дълга 2м., заема площ 2кв.м. Една плочка е широка 5м. и дълга 4м. има площ = 20кв.м. Площта която трябва да се покрие е 400 – 2 = 398 кв.м. Необходими са 398 / 20 = 19.90 плочки. Необходимото време е 19.90 * 0.2 = 3.98 минути.

Вход

От конзолата се четат 5 числа:

- **N** – дължината на страна от площадката в интервала [1...100]
- **W** – широчината на една плочка в интервала [0.1...10.00]
- **L** – дължината на една плочка в интервала [0.1...10.00]
- **M** – широчината на пейката в интервала [0...10]
- **O** – дължината на пейката в интервала [0...10]

Изход

Да се отпечата на конзолата две числа: броят плочки необходи за ремонта и времето за поставяне, всяко на нов ред.

Примерен вход и изход

Вход	Изход	Обяснения
20 5 4 1 2	19.9 3.98	Обща площ = 20 * 20 = 400; площ на пейката = 1 * 2 = 2 Площ за покриване = 400 – 2 = 398 Площ на плочки = 5 * 4 = 20 Необходими плочки = 398 / 20 = 19.9 Необходимо време = 19.9 * 0.2 = 3.98
40 0.8 0.6 3 5	3302.08333333333 660.416666666667	

Решение

```
namespace Repair.Models
{
    public class Model
    {
        private int n;
        public int N
```

```
{
    get { return n; }
    set
    {
        if (value > 0 && value < 101)
        {
            n = value;
        }
        else
        {
            throw new Exception("Incorrrect value");
        }
    }
}

private double w;
public double W
{
    get { return w; }
    set
    {
        if (value >= 0.1 && value <= 10.00)
        {
            w = value;
        }
        else
        {
            throw new Exception("Incorrrect value");
        }
    }
}

private double l;
public double L
{
    get { return l; }
    set
    {
        if (value >= 0.1 && value <= 10.00)
        {
            l = value;
        }
        else
        {
            throw new Exception("Incorrrect value");
        }
    }
}

private double m;
public double M
{
    get { return m; }
    set
    {
```

```
            if (value > 0 && value < 11)
            {
                m = value;
            }
            else
            {
                throw new Exception("Incorrrect value");
            }
        }
    }

    private double o;
    public double O
    {
        get { return o; }
        set
        {
            if (value > 0 && value < 11)
            {
                o = value;
            }
            else
            {
                throw new Exception("Incorrrect value");
            }
        }
    }

    public Model(int N, double W, double L, int M, int O)
    {
        this.N = N;
        this.W = W;
        this.L = L;
        this.M = M;
        this.O = O;
    }

    public Model() : this(0, 0, 0, 0, 0)
    {
        // Empty
    }

    public string CalculateResponse()
    {
        double area = n * n;
        area -= m * o;
        double tilesCount = area / (w * l);
        double time = tilesCount * 0.2;
        return $"{tilesCount}\n{time}";
    }
}

public class View
{
    public int N { get; set; }
```

```
        public double W { get; set; }
        public double L { get; set; }
        public int M { get; set; }
        public int O { get; set; }
        public string Response { get; set; }

        public View()
        {
            N = 0;
            M = 0;
            L = 0;
            W = 0;
            O = 0;
            Response = string.Empty;
            Input();
        }

        public void Input()
        {
            Console.Write("N = ");
            N = int.Parse(Console.ReadLine());
            Console.Write("W = ");
            W = double.Parse(Console.ReadLine());
            Console.Write("L = ");
            L = double.Parse(Console.ReadLine());
            Console.Write("M = ");
            M = int.Parse(Console.ReadLine());
            Console.Write("O = ");
            O = int.Parse(Console.ReadLine());
        }

        public void ShowResponse()
        {
            Console.WriteLine(Response);
        }
    }

    public class View
    {
        public int N { get; set; }
        public double W { get; set; }
        public double L { get; set; }
        public int M { get; set; }
        public int O { get; set; }
        public string Response { get; set; }

        public View()
        {
            N = 0;
            M = 0;
            L = 0;
            W = 0;
            O = 0;
            Response = string.Empty;
            Input();
```

```
        }

        public void Input()
        {
            Console.Write("N = ");
            N = int.Parse(Console.ReadLine());
            Console.Write("W = ");
            W = double.Parse(Console.ReadLine());
            Console.Write("L = ");
            L = double.Parse(Console.ReadLine());
            Console.Write("M = ");
            M = int.Parse(Console.ReadLine());
            Console.Write("O = ");
            O = int.Parse(Console.ReadLine());
        }

        public void ShowResponse()
        {
            Console.WriteLine(Response);
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var controller = new Controller();
        }
    }
}
```

Тема 2. Увод в концепцията за тестване. Писане на компонентни тестове

Задача 2.1. Тест на Axe

Заредете подадения solution наименуван Lab във Visual Studio.

Създайте следните тестове:

- Тествайте дали оръжието губи здравина след всяка атака
- Тествайте атака със счупено оръжие

Подсказки

Добавете нов проект наименуван Tests към решението (Solution), като при създаването на проекта изберете шаблон: NUnit Test Project.

Уверете се, че имате включен Reference към Lab проекта. Ако нямате добавете такъв чрез десен бутон върху проекта Tests -> [Add] -> [Reference] и от там изберете Solution:

Създайте клас AxeTests.

- Създайте метод за проверка на здравината

- По аналогичен начин създайте метод за проверка на атака със счупено оръжие

Изпълнете Build за целия solution, след което тестовете ще се покажат в Test Explorer прозорчето. Изпълнете ги. Резултатите от тестовете могат да бъдат видени в Test Explorer прозоречето.

Решение

```
using NUnit.Framework;

namespace Lab.Tests
{
    public class AxeTests
    {
        [Test]
        public void AxeLosesDurabilityAfterAttack()
        {
            // Arrange
            Axe axe = new Axe(10, 10);
            Dummy dummy = new Dummy(10, 10);

            // Act
            axe.Attack(dummy);

            // Assert
            Assert.AreEqual(9, axe.DurabilityPoints, "Axe durability does not change
after attack.");
        }

        [Test]
        public void BrokenAxeCantAttack()
        {
            // Arrange
            Axe axe = new Axe(2, 2);
            Dummy dummy = new Dummy(10, 10);

            // Act
            axe.Attack(dummy);
            axe.Attack(dummy);

            // Assert
            var e = Assert.Throws<InvalidOperationException>(() =>
axe.Attack(dummy));
            Assert.AreEqual(e.Message, "Axe is broken.");
        }

        [Test]
        public void CorrectAttackPoints()
        {
            // Arrange
            Axe axe = new Axe(1,1);

            // Act
            var points = axe.AttackPoints;
```

```
            // Assert
            Assert.AreEqual(1, points, "Incorrect attack points.");
        }
    }
}
```

Задача 2.2. Тест на Dummy

Създайте клас `DummyTests`.

Създайте следните тестове:

- Чучелото губи здраве, ако е атакувано
- Мъртво чучело хвърля изключение, ако е атакувано
- Мъртвото чучело може да даде XP
- Живото чучело не може да даде XP

Подсказки:

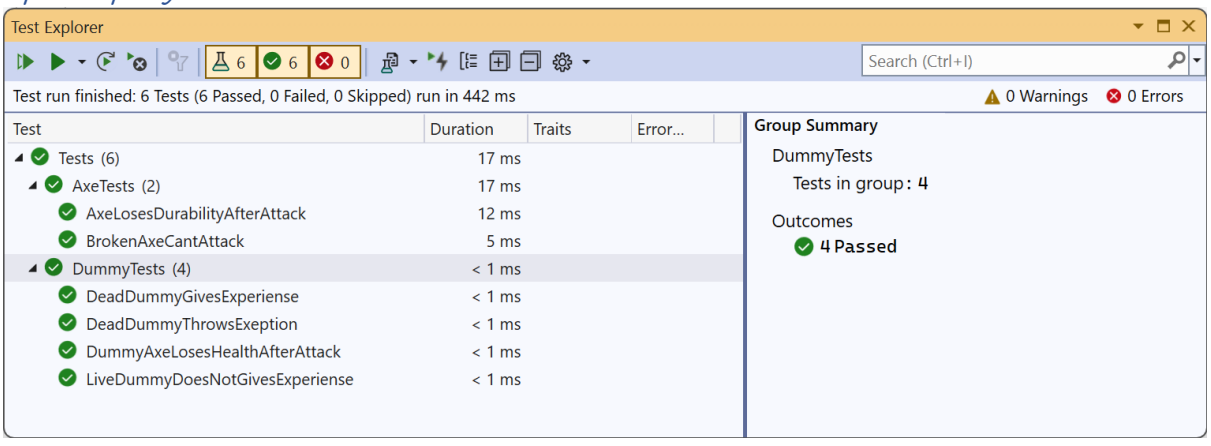
Следвайте логиката на предната задача

Решение

```
using NUnit.Framework;

namespace Lab.Tests
{
    public class DummyTests
    {
        [Test]
        public void DeadDummyGivesExperiense()
        {
            // Arrange
            Dummy dummy = new Dummy(1, 1);

            // Act
            dummy.TakeAttack(10);

            // Assert
            Assert.AreEqual(1, dummy.GiveExperience(), "Dummy Not Give Experience
After Attack");

        }

        [Test]
        public void DeadDummyThrowsException()
        {
            // Arrange
            Dummy dummy = new Dummy(1, 1);

            // Act
            dummy.TakeAttack(10);

            // Assert
            var e = Assert.Throws<InvalidOperationException>(() =>
dummy.TakeAttack(10));
```

```
            Assert.AreEqual(e.Message, "Dummy is dead.");
        }

        [Test]
        public void DummyLosesHealthAfterAttack()
        {
            // Arrange
            Axe axe = new Axe(1, 1);
            Dummy dummy = new Dummy(10, 10);

            // Act
            axe.Attack(dummy);

            // Assert
            Assert.AreEqual(9, dummy.Health, "Dummy Not Loses Health After Attack");
        }

        [Test]
        public void LiveDummyDoesNotGivesExperiense()
        {
            // Arrange
            Axe axe = new Axe(1, 100);
            Dummy dummy = new Dummy(10, 50);

            // Act
            axe.Attack(dummy);

            // Assert
            var ex = Assert.Throws<InvalidOperationException>(() =>
dummy.GiveExperience());
            Assert.AreEqual(ex.Message, "Target is not dead.");
        }

    }
}
```

Краен резултат:



Тема 3. Увод в концепцията за дебъгване. Откриване и отстраняване на проблеми

Задача 3.1. Множество инструкции

Напишете Компилатор с инструкции, който получава произволен брой инструкции. Програмата трябва да анализира инструкциите, да ги изпълни и изведе резултата. Трябва да бъде поддържан следния набор от инструкции:

- INC <операнд1> - увеличава операнд с 1
- DEC <операнд1> - понижава операнд с 1
- ADD <операнд1> <операнд2> - дава сбор на двата операнда
- MLA <операнд1> <операнд2> - дава произведението на двата операнда
- END – край на въвеждането

Изход

Резултатът на всяка инструкция трябва да бъде отпечатан на отделен ред на конзолата

Ограничения

- Операндите ще са валидни числа от [–2 147 483 648 … 2 147 483 647].

Тестове

Вход	Изход от програмата	Очакван изход
INC 0 END	**0** **0** **… (infinite)**	**1**
ADD 1323134 421315521 END	**422638655** **422638655** **… (infinite)**	**422638655**
DEC 57314183 END	**57314183** **57314183** **… (infinite)**	**57314182**
MLA 252621 324532 END	**379219748** **379219748** **… (infinite)**	**81983598372**

Решение

```
namespace InstructionSet
{
    public class Program
    {
        static void Main()
        {
            string opCode = Console.ReadLine();

            while (opCode != "END")
            {
```

```
            string[] codeArgs = opCode.Split(' ');

            long result = 0;
            switch (codeArgs[0])
            {
                case "INC":
                    {
                        int operandOne = int.Parse(codeArgs[1]);
                        result = ++operandOne;
                        break;
                    }
                case "DEC":
                    {
                        int operandOne = int.Parse(codeArgs[1]);
                        result = --operandOne;
                        break;
                    }
                case "ADD":
                    {
                        long operandOne = long.Parse(codeArgs[1]);
                        long operandTwo = long.Parse(codeArgs[2]);
                        result = operandOne + operandTwo;
                        break;
                    }
                case "MLA":
                    {
                        long operandOne = long.Parse(codeArgs[1]);
                        long operandTwo = long.Parse(codeArgs[2]);
                        result = (long)(operandOne * operandTwo);
                        break;
                    }
            }

            Console.WriteLine(result);

            opCode = Console.ReadLine();
        }
    }
}
```

Задача 3.2. Положителен

Вие ще получите няколко последователности от числа на конзолата; Вашата задача е да премахнете всички отрицателни числа и отпечатате обратно всяка поредица.

На първия ред на въвеждане ви се дават брой N – брой на последователностите.

На всеки от следващите N реда ще получите няколко числа, заобиколени от интервали.

Вие трябва да проверите всяко число, ако то е положително – го отпечатвате на конзолата; ако то е отрицателно – добавяте неговата

стойност към стойността на следващото число и извеждате резултата ако той не е отрицателен. Изпълнявате събиране само веднъж, например, ако имате последователността: -3, 1, 3, алгоритъмът ще е както следва:

- -3 е отрицателно => го добавяме към следващото число (1) => -3 + 1 = -2 все още отрицателно => не се печата нищо (и не запазваме сбора, спираме тук).
- Следващото число, считаме, че е 3, което е положително => го отпечатваме.

Ако не могат да бъдат получени числа по този начин за дадена последователност, печатаме "(empty)".

Пример

Вход	Очакван изход	Коментари
3 3 -4 5 2 123 -1 -1 3 4 -2 1	3 1 2 123 3 4 (empty)	(3) (-4 + 5 = 1 > 0) (2) (123) (-1 + (-1) < 0) (3) (4) (-2 + 1 < 0)

Изход

Печата на конзолата всяка променена последователност на отделен ред

Ограничения

- Числото N ще е цяло в интервала [1 ... 15].
- Числата в последователността ще са цели в интервала [-1000 ... 1000].
- Броят на числата във всяка последователност ще са в интервала [1 ... 20].
- Може да има интервали на всякъде между числата в дадената последователност

Тестове

Вход	Програмен изход	Очакван изход
3 3 -4 5 2 123 -1 -1 3 4 -2 1	**Exception…**	**3 1 2 123** **3 4** **(empty)**
1 0 -2 2 -2 3	**Exception…**	**0 0 1**

Решение

```
namespace BePositive
{
    internal class Program
    {
        public static void Main()
        {
            int countSequences = int.Parse(Console.ReadLine());

            for (int i = 0; i < countSequences; i++)
            {
                string[] input = Console.ReadLine().Trim().Split(' ');
                var numbers = new List<int>();

                for (int j = 0; j < input.Length; j++)
                {
                    if (!input[j].Equals(string.Empty))
```

```
                {
                    int num = int.Parse(input[j]);
                    numbers.Add(num);
                }
            }

            bool found = false;

            for (int j = 0; j < numbers.Count; j++)
            {
                int currentNum = numbers[j];

                if (currentNum >= 0)
                {
                    if (found)
                    {
                        Console.Write(" ");
                    }

                    Console.Write(currentNum);

                    found = true;
                }
                else
                {
                    currentNum += numbers[j + 1];

                    if (currentNum >= 0)
                    {
                        if (found)
                        {
                            Console.Write(" ");
                        }

                        Console.Write(currentNum);

                        found = true;
                    }

                    j++;
                }
            }

            if (!found)
            {
                Console.WriteLine("(empty)");
            }
        }
    }
}
```

Задача 3.3. Array Test

Получавате едно число n, представляващо размера на масив от цели числа и на следващия ред елементите на масива, разделени с интервали. След това, ви се предоставя произволен брой команди във формат: "[действие] [i-тия елемент] [стойност]". За край на командите се използва "stop".

Действията могат да са "multiply", "add", "subtract", "rshift" or "lshift". Действията "multiply", "add" и "subtract" имат параметри. Първият параметър е броят на елементите, които трябва да бъдат променени. Вторият параметър е стойността, с която ние манипулираме елемента. The command "lshift" iterates through the array changing each element's position with 1 to the left. The first element which should go outside the array will eventually become last. E.g. {1, 2, 3} "lshift" will become {2, 3, 1}. The command "rshift" does the same thing but changes the positions with 1 to the right. The last element which should go outside the array, becomes first. E.g. {1, 2, 3} "rshift" will become {3, 1, 2}.

Командата "lshift" циклично променя позицията на всеки елемент в масива с 1 отляво. Първият елемент, който трябва да излезе извън масива ще стане последен. Например {1, 2, 3} "lshift" ще стане {2, 3, 1}. Командата "rshift" прави същото нещо, но променя позициите с 1 вдясно. Последният елемент, който трябва да излезе извън масива, става първи. Например {1, 2, 3} "rshift" ще стане {3, 1, 2}.

Пример:

```
5
3 0 9 333 11
add 2 2
subtract 1 1
multiply 3 3
stop
```

Ние изместваме всеки 2ри елемент на дясно два пъти. След преместване получаваме масив {2 2 27 333 11}.

Изход

За всяко действие изведете елементите на масива на нов ред на конзолата.

Ограничения

- Числото n ще бъде цяло в интервала[1 ... 15].
- Всеки елемент на масива ще е цяло число в интервала [0 ... 2^{63}-1].
- Числото i и броят на командите ще бъде цяло число в интервала [1 ... 10].
- Стойността на числото ще бъде цяло число в интервала [-100 ... 100]. Ако командата е "rshift" или "lshift" няма да има параметри.

Тестове

Вход	Програмен изход	Очакван изход

5 3 0 9 333 11 add 2 2 subtract 1 1 multiply 3 3 stop	3 0 9 333 11 3 0 9 333 11	3 2 9 333 11 2 2 9 333 11 2 2 27 333 11

Решение

```
namespace ArrayTest
{
    public class Program
    {
        private const char ArgumentsDelimiter = ' ';

        public static void Main()
        {
            int sizeOfArray = int.Parse(Console.ReadLine());

            long[] array = Console.ReadLine()
                .Split(ArgumentsDelimiter)
                .Select(long.Parse)
                .ToArray();

            string[] command =
Console.ReadLine().Split(ArgumentsDelimiter).ToArray();

            while (!command[0].Equals("stop"))
            {
                int[] args = new int[2];

                if (command[0].Equals("add") ||
                    command[0].Equals("subtract") ||
                    command[0].Equals("multiply"))
                {
                    args[0] = int.Parse(command[1]);
                    args[1] = int.Parse(command[2]);
                }

                array = PerformAction(array, command[0], args);

                PrintArray(array);

                command = Console.ReadLine().Split(ArgumentsDelimiter).ToArray();
            }
        }

        static long[] PerformAction(long[] arr, string action, int[] args)
        {
            long[] array = arr.Clone() as long[];
            int pos = args[0] - 1;
            int value = args[1];

            switch (action)
            {
```

```
                case "multiply":
                    array[pos] *= value;
                    break;
                case "add":
                    array[pos] += value;
                    break;
                case "subtract":
                    array[pos] -= value;
                    break;
                case "lshift":
                    array = ArrayShiftLeft(array);
                    break;
                case "rshift":
                    array = ArrayShiftRight(array);
                    break;
            }

            return array;
        }

        private static long[] ArrayShiftRight(long[] array)
        {
            long movingElement = array[array.Length - 1];
            for (int i = array.Length - 1; i >= 1; i--)
            {
                array[i] = array[i - 1];
            }
            array[0] = movingElement;
            return array;
        }

        private static long[] ArrayShiftLeft(long[] array)
        {
            long movingElement = array[0];
            for (int i = 0; i < array.Length - 1; i++)
            {
                array[i] = array[i + 1];
            }
            array[array.Length - 1] = movingElement;
            return array;
        }

        private static void PrintArray(long[] array)
        {
            Console.WriteLine(string.Join(" ", array));
        }
    }
}
```

Задача 3.4. Подстринг

Даден е text и число j. Вашата задача е да потърсите в текста буква 'p' (ASCII код 112) и изведете 'p' и j букви от него надясно.

Например: Даден е текста "phahah put" и число 3. Ние намираме съвпадение на 'p' в първата буква, така че извеждаме 'p' и три букви надясно. Резултатът е "phah". Ние продължаваме да търсим друго съвпадение и намираме 'p', но няма 3 букви след него надясно, така отпечатваме само "put".

Всяко съвпадение трябва да се отпечатва на отделен ред. Ако няма съвпадения на 'p' в текста, ние печатаме "no".

Изход

Всяко съвпадение трябва да се отпечатва на отделен ред. Ако няма съвпадения на 'p' в текста, ние печатаме "no".

Ограничения

- Числото j трябва да е в интервала [0 ... 100].

Тестове

Вход	Програмен изход	Очакван изход
phahah put 3	no	phah put
No match 5	no	no
preparation 4	no	prepra
preposition 0	no	p p

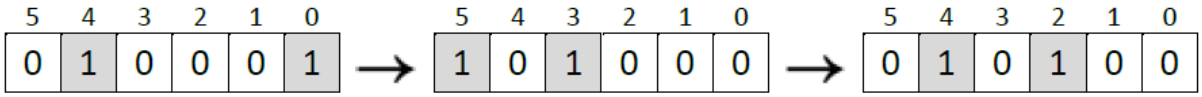
Решение

```
namespace Substring
{
    internal class Program
    {
        public static void Main()
        {
            string text = Console.ReadLine();
            int jump = int.Parse(Console.ReadLine());

            // fix: p (bg) => p (en)
            const char Search = 'p';
            bool hasMatch = false;

            for (int i = 0; i < text.Length; i++)
            {
                if (text[i] == Search)
                {
                    hasMatch = true;

                    // fix: string len fixes
                    int endIndex = jump + 1;
                    if (i + endIndex >= text.Length)
                    {
```

```
                    endIndex = text.Length - i;
                }

                string matchedString = text.Substring(i, endIndex);
                Console.WriteLine(matchedString);
                i += jump;
            }
        }

        if (!hasMatch)
        {
            Console.WriteLine("no");
        }
    }
}
```

Задача 3.5. Малка въртележка

Дадено Ви е число **n**, **брой измествания** и **посоки**. Програмата трябва да измества битовете в таблица с 6 **клетки**. Изместването трябва да придвижи всички битове с **1позиция** в дадената посока. Например ни се дават числото **17** и **два пъти** да се измести "**right**".



Забележка: Ако бит излиза извън таблицата, то следва да започнете отначало от другия край.
Резултатът е 20.

Изход

Резултатното число (след като са направени всички измествания) трябва да се изведе на екрана.

Ограничения

- Числото **n** ще бъде в интервала [0 ... 63].

Тестове

Вход	Програмен изход	Очакван изход
32 2 right right	**8**	**8**
63 1 left	**126**	**63**
59 4	**183**	**62**

left left left left		
45 3 left right left	**90**	**27**

Решение

```
namespace BitCarousel
{
    internal class Program
    {
        static void Main()
        {
            byte number = byte.Parse(Console.ReadLine());
            byte rotations = byte.Parse(Console.ReadLine());

            for (int i = 0; i < rotations; i++)
            {
                string direction = Console.ReadLine();

                if (direction == "right")
                {
                    int rightMostBit = number & 1;
                    number >>= 1;
                    number |= (byte)(rightMostBit << 5);
                }
                else if (direction == "left")
                {
                    int leftMostBit = (number >> 5) & 1;
                    number <<= 1;
                    number &= 63;
                    number |= (byte)leftMostBit;
                }
            }

            Console.WriteLine(number);
        }
    }
}
```

Тема 4. Преработка на кода и постепенни промени

Задача 4.1. Доброто име сложни програми поправя

Прегледайте програмния файл ArraySlider.docx и ArraySlider.cs. Това е истинска изпитна задача и примерно решение, което е абсолютно вярно и изкарва 100/100 точки в системата judge. Въпреки верността си, кодът е зле написан и труден за разбиране и поддръжка. Например, всички

променливи са наименувани с една буква и, въпреки че кодът е сравнително къс, хората лесно се губят в него.

Вашата задача: Дайте смислени имена на променливите

Общи съвети:

- Използвайте английски! Дори и да е само парче код, което вие и само вие ще четете, просто използвайте английски. Без извинения.
- Избягвайте съкращения: dateTimeFormatter е по-добре от dtf или dtFormatter
- Престараването също е лоша идея.

Пример:
Като гледаме кода, забелязваме реда var r = Console.ReadLine(); Очевидно е някакъв вид вход, но не можем да разберем какъв точно, какво прави и защо е там. Описанието на задачата ни казва, че след първия ред код ще получим допълнителни команди, всяка на нов ред. Можем да заменим реда с: string command = Console.ReadLine(); string[] commandTokens = Console.ReadLine().Split(); също е допустимо, в зависимост от това как решите да структурирате кода си.

Бонус задача: Някои части от кода са дублирани или ненужни. Намерете начин да направите кода по-сбит и по-елегантен.

Решение
вж. ArraySLider

Задача 4.2. LINQ заявки

Дадено ви е приложението Orders, което обработва поръчки. Приложението има продукти, категории и поръчки. Всеки артикул има уникално ID. Елементите (продукти, категории и поръчки) са свързани чрез своите ID-та. Например, ако има следните категории:

```
ID: 1, Name: Beverages
ID: 2, Name: Condiments
```

и следните продукти:

```
ID: 1, Name: Chai, CategoryID: 1
ID: 4, Name: Chef Anton's Cajun Seasoning, CategoryID: 2,
```

това значи, че продуктът Chai се съдържа в категорията Beverages.

Дадена ви е програма, която чете отделени със запетая текстови файлове и зарежда информацията в системата. След това програмата изпълнява четири LINQ заявки, както следва:

- Намира имената на петте най-скъпи продукта
- Намира броя продукти във всяка категория
- Намира петте топ продукта (с максимален брой поръчки)

- Намира името на най-доходната категория (тази, която генерира най-голям общ доход)

Програмата работи вярно. Задачата ви е да промените имената на всички идентификатори, които трябва да бъдат преименувани, за да направите кода по-четлив, по-ясен за разбиране и по-лесен за промяна и разширение.

Може да преправите каквото искате (да извадите методи / класове, да добавяте коментари и т.н.). Само се уверете, че програмата все още работи вярно след промените ви.

Решение
вж. Orders

Задача 4.3. Непознат метод

Даден ви е проектът **ConsoleApplication1**, който е верен и няма бъгове, но кодът е написан много зле. Преименувайте всички идентификатори, които трябва, за да направите кода по-четим и по-лесен за разбиране и многократно използване.

Може да се наложи да дебъгнете и да разгледате програмата по-отблизо.

Решение
вж. MatricMultiplication

Задача 4.4. Игра

Даден ви е проект, наречен **Application2**. Преименувайте всички идентификатори (имена на променливи, методи, класове, пространства от имена и т.н.), които е нужно, за да направите кода по-четлив и по-лесен за разбиран и за многократна употреба.

Преправете и друго в проекта, не само имената, така че кодът да е добре форматиран и лесен за четене, разбиране и промяна.

Идеи: може да добавяте коментари, за да обясните някои по-сложни места в кода; може да подобрите форматирането на кода; може да извлечете код в методи, класове и пространства от имена; може да приложите ООР принципите и други.

Решение
вж. Minesweeper

Задача 4.5. Форматиране на нелепо лош код

Дадено ви е C# решение, което съдържа няколко интерфейса и класа, изграждащи проста игра „Змия“. Логиката е добра, но форматирането на кода е абсолютно отвратително.

Вашата задача е да го поправите.

Опрете се на досегашния си опит и общата си култура. Ако искате, можете да прегледате и основните правила за добър стил на форматиране на кода.

Няколко съвета:

- Логически отделните блокове код трябва да са отделени и с празен ред.
- Методите, конструкторите и свойствата трябва да са отделени.
- Не слагайте паузи там, където не са необходими.
- Фигурните скоби трябва да са на нов ред.

Решение

вж. YoloSnake

Задача 4.6. Разширения на низове

Даден ви е файл с програмен код от Visual Studio, наречен StringExtensions.cs. Създайте XML документация за файла, всички класове и техните членове. Добавете подходящи коментари в кода на местата, където това би подобрило четливостта и разбирането му.

Решение

вж. StringExtensions

Задача 4.7. Вмъкване навътре

Даден ви е C# код за задачата Five Special Letters. Кодът е верен, но много труден за четене и разбиране.

Преправянето на цялото това великолепие няма да е лесно, така че се фокусирайте върху цикъла for, който започва на ред 67. Скоро ще забележите, че нивото навмъкване е далеч над препоръчителното.

Вашата задача е да намерите начин да намалите броя на if-else блоковете. Обмислете извличането на повтарящата се логика в метод. Променете цикъла for в някой по-подходящ.

Решение

вж. FiveSpecialLetters

Задача 4.8. Преправяне на методи

Използвайте VS решението "Methods.sln".

- Преправете кода така, че да следва насоките за висококачествени методи.
- Уверете се, че обработвате грешките коректно: ако методите не могат да изпълнят това, което подсказва името им, хвърлете изключение (не връщайте грешен резултат).
- Подсигурете силна специализация и слаба зависимост, добро именуване, никакви странични ефекти и т.н.

Решение

вж. Methods

Задача 4.9. Абстракция

Използвайте VS решението "Abstraction".

- Поправете кода му, за да има добра абстракция.
- Преместете свойствата и методите от класа Figure на правилните им места.
- Преместете общите методи в тялото на базовия клас.
- Махнете всичкия повтарящ се код (свойства / методи / друг код).

Използвайте VS решението "Abstraction".

- Осигурете добро капсулиране във всички класове.
- Уверете се, че във вътрешното състояние на класовете не могат да бъдат зададени неверни стойности.

Решение

вж. Abstractions

Задача 4.10. Специализация и зависимост

Използвайте VS решението "Cohesion-and-Coupling".

- Поправете кода му така, че да следва принципите на добра абстракция, слаба зависимост и силна специализация.
- Разделете класа Utils на други класове, които имат силна специализация и са слабо зависими вътрешно.

Решение

вж. Cohesion and Coupling

Задача 4.11. Наследяване и полиморфизъм

Използвайте VS решението "Inheritance-and-Polymorphism"

- Следвайте най-добрите практики за висококачествен програмен код.
- Проектирайте наново класовете и пренапишете кода.
- Извлечете абстрактен базов клас и преместете в него всички общи свойства.
- Капсулирайте полетата и се уверете, че задължителните полета не са оставени без стойност.
- Използвайте отново (reuse) дублирания код чрез методи в базовия клас.

Решение

вж. Inheritance and Polymorphism

Задача 4.12. Изчистете лошия код

Използвайте решението "Matrica.sln" в Visual Studio. Даден ви е файлът "Rotating-Walkin-in-Matrix.docx", който обяснява задачата на програмата. Подобрете вътрешното качество на този проект. Може да следвате тези стъпки:

1. Направете някои начални преработки, като:
   - Преформатиране на кода.
   - Преименуване на зле именуваните променливи.
2. Направете така, че кодът да може да се тества.
   - Помислете как да тествате конзолно-базиран вход / изход.
3. Напишете компонентни тестове (unit tests). Оправете всички бъгове, които откриете докато тествате.
4. Преправете кода, като следвате насоките от този урок. Направете го стъпка по стъпка. Изпълнявайте тестовете след всяка голяма промяна.

Решение

вж. Rotating Walk in Matrix

Тема 5. Инструменти за разработка

Задача 5.1. Дефиниране на класа BankAccount

Създайте клас **`BankAccount`**.

Класът трябва да има свойства:

- `ID: int`
- `Balance: double`

Класът трябва да има и следния конструктор:

- `BankAccount(int id, double balance)`

Решение

```
namespace BankAccount
{
    public class BankAccount
    {
        private int id;

        public int Id
        {
            get { return id; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Must be positive.");
                }
                id = value;
```

```
            }
        }

        private decimal balance;

        public decimal Balance
        {
            get { return balance; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Must be positive.");
                }
                balance = value;
            }
        }

        public BankAccount() : this(0,0)
        {
           // empty
        }

        public BankAccount(int id, decimal balance)
        {
            this.Id = id;
            this.Balance = balance;
        }

        public override string ToString()
        {
            return string.Format($"Id: {this.Id}, Balance: {this.Balance}");
        }
    }
}
```

Задача 5.2. Дефиниране на класа Person

Създайте клас Person (или използвайте вече създадените класове от предните уроци).

Класът трябва да има свойства за име, възраст и банкови сметки:

- Name: string
- Age: int
- Accounts: `List<BankAccount>`

Класът трябва да има и следния конструктор:

- `Person(string name, int age, List<BankAccount> accounts)`

Класът трябва да има и public метод за:

- `GetBalance(): double`

Решение

```
namespace Person
{
    public class BankAccount
    {
        private int id;

        public int Id
        {
            get { return id; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Must be positive.");
                }
                id = value;
            }
        }

        private decimal balance;

        public decimal Balance
        {
            get { return balance; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Must be positive.");
                }
                balance = value;
            }
        }

        public BankAccount() : this(0,0)
        {
           // empty
        }

        public BankAccount(int id, decimal balance)
        {
            this.Id = id;
            this.Balance = balance;
        }

        public override string ToString()
        {
            return string.Format($"Id: {this.Id}, Balance: {this.Balance}");
        }
    }

    public class Person
    {
```

```
private string name;

public string Name
{
    get { return name; }
    set
    {
        if (string.IsNullOrEmpty(value))
        {
            throw new ArgumentException("Name can not be null or empty.");
        }
        name = value;
    }
}

private int age;

public int Age
{
    get { return age; }
    set
    {
        if (age >= 18 && age <= 100)
        {
            throw new ArgumentException("Age must be from 18 to 100");
        }
        age = value;
    }
}

private List<BankAccount> accounts;

public List<BankAccount> Accounts
{
    get { return accounts; }
    set
    {
        if (value == null && value.Count() > 0)
        {
            throw new ArgumentException("Collection must not be empty.");
        }
        accounts = value;
    }
}

public Person() : this("Noname", 18, new List<BankAccount>())
{
    // empty
}

public Person(string name, int age, List<BankAccount> accounts)
{
    this.Name = name;
    this.Age = age;
    this.Accounts = accounts;
```

```
        }

        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();

            sb.Append(string.Format($"Name: {this.Name}") + Environment.NewLine);
            sb.Append(string.Format($"Age: {this.Age}") + Environment.NewLine);
            sb.Append(string.Format($"Bank Accounts:") + Environment.NewLine);
            Accounts.ForEach(account => sb.Append(string.Format($"\t{account}") +
Environment.NewLine));

            return sb.ToString();
        }
    }
}
```

Тема 6. Пакети и външни библиотеки

Тема 6. Пакети и външни библиотеки

Задача 6.1. Продукти

Създаване на клас Product

Създайте съвсем обикновен клас Product със свойства за:

- Id (int) – номер на продукта
- Name (string) – име на продукта
- Price (decimal) – цена на продукта
- Stock(int) – наличност на продукта
- Expiry (DateTime) – срок на годност на продукта

Сериализиране на обект

Първо ще създадем един обект от клас Product. След това използвайки библиотеката ще извършим сериализация на обекта, т.е. да го превърнем в JSON низ.

За да може да ползваме класа JsonConvert, ще трябва да добавим и using Newtonsoft.Json; в началото на Program.cs

Десериализиране на JSON

Сега нека да видим как работи и обратния процес, ще десериализираме JSON низ, в който е описан масив от размери на дрехи. За целта ще работим с класа JArray, за който ще имате нужда също от using Newtonsoft.Json.Linq; в горната част на Program.cs. Кодът десериализира низа, превръщайки го в масив, на който можем да приложим foreach или произволна LINQ операция.

Решение

```
namespace Products
{
    public class Product
```

```
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public decimal Price { get; set; }

        public int Stock { get; set; }

        public DateTime Expiry { get; set; }

        public Product
        (
            int Id,
            string Name,
            decimal Price,
            int Stock,
            DateTime Expiry
        )
        {
            this.Id = Id;
            this.Name = Name;
            this.Price = Price;
            this.Stock = Stock;
            this.Expiry = Expiry;
        }

        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();

            sb.Append($"Id: {Id}, ");
            sb.Append($"Name: {Name}, ");
            sb.Append($"Price: {Price}, ");
            sb.Append($"Stock: {Stock}, ");
            sb.Append($"Expiry: {Expiry} ");
            sb.Append(Environment.NewLine);

            return sb.ToString();
        }
    }

    public class Program
    {

        static void Main(string[] args)
        {
            // Step 1. Serialize Object to JSON
            Product product = new Product(1,"Coca Cola", 2.4m, 100, new
DateTime(2023,09,10));
            string json = JsonConvert.SerializeObject(product);

            Console.WriteLine("Step 1. Serialize Object to JSON");
            Console.WriteLine(json);
            Console.WriteLine(Environment.NewLine);
```

```
            // Step 2. Deserialize JSON to Object
            Product second = JsonConvert.DeserializeObject<Product>(json);

            Console.WriteLine("Step 2. Deserialize JSON to Object");
            Console.WriteLine(second);
            Console.WriteLine(Environment.NewLine);
        }
    }
}
```

Задача 6.2. Оптично разпознаване на символи

Инсталиране на Tessearct чрез NuGet

Сега трябва да инсталираме Tesseract библиотеката чрез NuGet. За целта:

- Натиснете Ctrl + Q, за да използвате Quick Launch
- Въведете nuget
- Изберете Tools -> Library Package Manager -> Manage NuGet Packages for Solution...
- Намерете Tesseract и инсталирайте.

Добавяне на файлове за разпознаване

Tessearct е библиотека за разпознаване на текст от изображение - Optical Character Recognition (OCR). Няма да навлизаме в техническите детайли на това как работи тази технология, а ще се концентрираме върху използването на библиотеката за тази нейна цел. За да разпознава успешно отделни знаци, Tessearct се нуждае от модел с данни. Ще използваме готов набор от данни. Файловете на модела трябва да бъдат поставени там където се създава .exe файла на приложението. В случая файла се намира в bin/debug папката на проекта. Там трябва да поставите и папката tessdata, която е предоставена като допълнителен ресурс.

Добавяне на изображение за разпознаване

Файлът с изображението test.png трябва се намира в bin/debug папка tessdata на проекта.

Прочитане на текста от изображението

Програмният код, с който можем да извършим прочитане е изненадващо прост. Трябва да си създадем един низ, в който да запишем пълния път към файла и името му. След това създаваме обект от клас TesseractEngine, указвайки езика, на който е текста, както и името на папката с данните.

След това създаваме обект за изображението, а накрая чрез метода Process получаваме и обект за страницата – този обект има метод GetText(), който съдържа нашия текст.

Ще се задоволим на този етап с този резултат, но ще добавим, че все пак той може да бъде подобрен по редица начини, например:

- Допълнителна обработка на изображението, чрез методи и класове от библиотеката на Tesseract или външен софтуер
- Допълнително или по-добро трениране на данните на модела на Tesseract.

Решение

```
namespace OpticalCharacterRecognition
{
    internal class Program
    {
        static void Main(string[] args)
        {
            string fileName = AppContext.BaseDirectory + "tessdata\\test.png";
            using (var engine = new TesseractEngine(@"tessdata", "eng"))
            {
                using (var image = Pix.LoadFromFile(fileName))
                {
                    using (var page = engine.Process(image))
                    {
                        string text = page.GetText();
                        Console.WriteLine(text);
                    }
                }
            }
        }
    }
}
```

Задача 6.3. Вицове за Чък Норис

Напишете приложение което изтегля вицове за Чък Норис, като използвате JSON и API, достъпен в Интернет на адрес: https://api.chucknorris.io/jokes/random

Решение

```
namespace ChuckNorrisJokes
{
    public class Joke
    {
        public List<object> categories { get; set; }
        public string created_at { get; set; }
        public string icon_url { get; set; }
        public string id { get; set; }
        public string updated_at { get; set; }
        public string url { get; set; }
        public string value { get; set; }
    }

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

```
            // Step 1. Get Chuck Norris Joke from Internet
            string json = GetJoke().Result;

            // Step 2. Deserilize JSON to Object
            Joke joke = JsonConvert.DeserializeObject<Joke>(json);

            // Step 3. Print Joke
            Console.WriteLine(joke.value);
        }

        private static async Task<string> GetJoke()
        {
            string url = "https://api.chucknorris.io/jokes/random";
            HttpClient client = new HttpClient();
            return await client.GetStringAsync(url);
        }
    }
}
```

Тема 7. Свързване на приложения с бази от данни

Задача 7.1. Minions

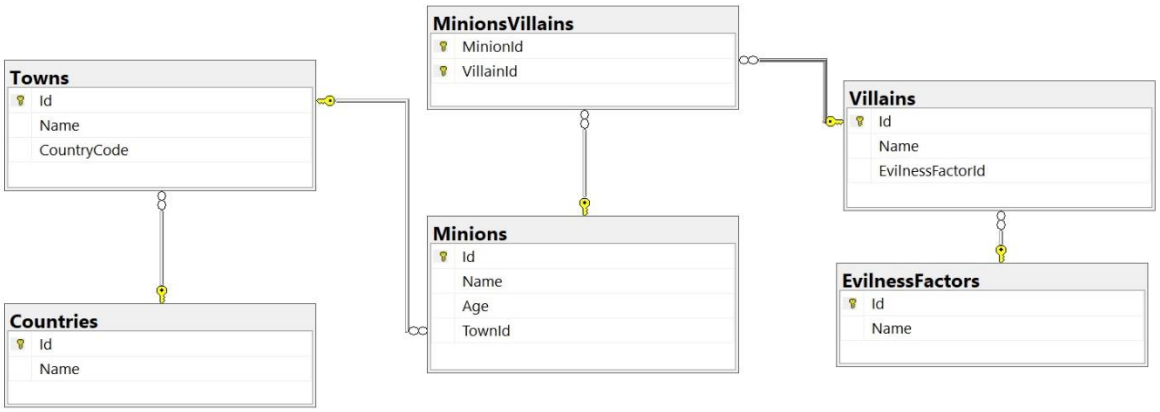
Първоначална настройка

Напишете програма, която се свързва с вашия локален сървър localhost. Създайте нова база данни наречена MinionsDB където ще пазим информация за нашите миниони и злодеи.

За всеки миньон трябва да пазим информация за неговото име, възраст и град. За всеки град има информация за държавата, в която се намира. Злодеите има име и фактор на злина (супер добър, добър, лош, зъл, супер зъл). Всеки миньон може да служи на няколко злодея и всеки злодей може да има няколко миньона, които му служат. Запълнете таблиците с поне по 5 записа във всяка.

Накрая трябва да имат следните таблици:

- Countries
- Towns
- Minions
- EvilnessFactors
- Villains
- MinionsVillains



Имена на злодеи

Напишете програма, която отпечатва на конзолата имената на всички злодеи и броят на миниона, на тези от тях, които имат по повече от 3 миниона, като подредбата е в намалящ порядък по броя на миниона.

Пример

Изход
Gru - 6 Victor - 4 Jilly – 4

Имена на миниони

Напишете програма, която отпечатва на конзолата всички имена на миниони и възраст по даден злодейски id, подредени по име в азбучен ред.
Ако няма злодей с даденото ID, изведете "No villain with ID <VillainId> exists in the database."
Ако съответният злодей няма миниони, изведете "(no minions)" на втори ред.

Пример

Вход	Изход
1	Villain: Gru 1. Bob 13 2. Kevin 14 3. Steward 19

Вход	Изход
3	Villain: Victor 1. Bob 13 2. Simon 22

Вход	Изход
2	Villain: Victor Jr. (no minions)

Вход	изход
10	*No villain with ID 10 exists in the database.*

Добавете минион

Напишете програма, която прочита информация за миниона и неговият злодей и добавя този миниона в базата данни. В случай че, градът на миниона не е в БД, трябва да го въведете също. В случай че, злодеят не

съществува в базата данни, добавете и него със стойност по подразбиране за фактор на злина "evil". Накрая задайте новият миньон да бъде слуга на злодея. Извеждайте подходящи съобщения след всяка операция.

Вход

Входът е на два реда:

- На първи ред, ще получите информация за миньон във формат "Minion: <Name> <Age> <TownName>"
- На следващия – злодейска информация във формат "Villain: <Name>"

Изход

След завършване на операцията, трябва да изведете едно от следните съобщения:

- При добавяне на нов град: "Town <TownName> was added to the database."
- При доабвяне на нов злодей: "Villain <VillainName> was added to the database."
- Накрая, след успешно добавяне на миниона към БД и задаването му като слуга: "Successfully added <MinionName> to be minion of <VillainName>."

Бонус задача: Уверете се, че всички операции се изпълняват успешно. В случай на грешка не променяйте БД.

Пример

Вход	Изход
`Minion: Bob 14 Berlin` `Villain: Gru`	`Successfully added Robert to be minion of Gru.`
`Minion: Cathleen 20 Liverpool` `Villain: Gru`	`Town Liverpool was added to the database.` `Successfully added Cathleen to be minion of Gru.`
`Minion: Mars 23 Sofia` `Villain: Poppy`	`Villain Poppy was added to the database.` `Successfully added Mars to be minion of Poppy.`

`Minion: Carry 20 Eindhoven`
`Villain: Jimmy`
	`Town Eindhoven was added to the database.`
	`Villain Jimmy was added to the database.`
	`Successfully added Carry to be minion of Jimmy.`

Промяна на регистъра на имената на градовете

Напишете програма, която променя всички имена на градове към главни букви за дадена държава.

Ще получите един ред вход с името на държавата.

Изведете броят на градовете, които са променени във формат "<ChangedTownsCount> town names were affected.". На втори ред, изведете имената, които са били променени, орделени със запетая и интервал.

Ако не са променени градове (няма градове свързани към държавата), изведете "No town names were affected.".

Пример

Вход	Изход
Bulgaria	3 town names were affected. [SOFIA, VARNA, BURGAS]
Germany	No town names were affected.

Премахване на злодей

Напишете програма, която получава ID на злодей, изтрива го от БД и освобождава неговите миньони. Изведете два реда името на изтрития злодей във формат "<Name> was deleted." И броят на миньоните, които са освободени във формат "<MinionCount> minions were released.". Уверете се, че всички операции са минали както е планирано, в противен случай не извършвайте промени по БД.

Ако няма злодей в БД с даденото ID, изведете "No such villain was found.".

Пример

Вход	Изход
1	Gru was deleted. 6 minions were released.
3	Victor was deleted. 0 minions were released.
101	No such villain was found.

Изведете всички имена на миньони

Напишете програма, която извежда имената на всички миньони от таблицата с миньони в следния порядък: първи запис, последен запис, първи + 1, последен - 1, първи + 2, последен - 2 ... първи + n, последен - n.

1	10	2	9	3	8	4	7	5	6

Пример

Оригинален ред	Изход
Bob Kevin Steward Jimmy Vicky Becky Jully	Bob Jully Kevin Becky Steward Vicky Jimmy

Увеличение на възраст на миньон

Прочете от конзолата миньонски IDта разделени с интервал. Увеличете възрастта на тези миньони с 1 и направете техните имена с първа главна буква. Накрая, изведете името и възрастта на всички миньони в БД, всеки на нов ред във формат "<Name> <Age>".

Пример

Minions

Id	Name	Age
1	bob	14
2	stuart	22
3	kevin	13
4	jimmy	49
5	vicky jackson	26

Вход	Изход
2 1 4	Bob 15 Stuart 23 kevin 13 Jimmy 50 vicky jackson 26

Вход	Изход
5	bob 14 stuart 22 kevin 13 jimmy 49 Vicky Jackson 27

Задача 7.2. Да напишем ORM
Вж. MiniORM

Задача 7.3. Да ползваме ORM
Вж. MiniORM.DEmo

Задача 7.4. Да направим CRUD приложение без ORM
Вж. CrudWithoutOrm

Задача 7.5. Да направим CRUD приложение с ORM
Вж. CrudWithOrm

Тема 8. Създаване на приложения с няколко потребителски интерфейса

Създаване на просто приложение

В рамките на това упражнение ще направим три различни интерфейса за нашето приложение: конзолен, уеб и десктоп. Това упражнение се явява обобщение на упражненията досега.

Създаване на проектите и библиотеките от класове

Започнете със създаване на следните проекти и класове от библиотеки в Solution:

- ASP.NET Web Application –WebApp – уеб интерфейс
- Windows Forms Application – WinFormsApp – десктоп интерфейс
- Console Application – ConsoleApp – конзолен интерфейс
- Class Library – Data – слой за данни
- Class Library – Business – слой за услуги

След като сте създали всичките проекти, инсталирайте **Entity Framework Core** за целия solution.

Бази Данни и референции

Добавете следните референции:

- Business – референция към Data
- WebApp – референция към Business и Data
- WinFormsApp – референция към Business и Data
- ConsoleApp – референция към Business и Data

След това за всеки от проектите добавете в съответния конфигурационен файл низовете за връзка – вие вече знаете как да стане това.

Реализиране на интерфейсите

Всеки един от интерфейсите тук се реализира по абсолютно същия начин, по който и в предните упражнения. Реализирайте интерфейсите и изпълнете Build за целия solution.

Превключване на интерфейсите

Ако се опитате да стартирате приложението през Visual Studio, то ще стартира през един от интерфейсите. За да превключите, отидете на съответния проект, цъкнете с десен бутон и изберете Set as StartUp Project. По този начин чрез Visual Studio ще може да стартирате приложението с различни интерфейси.

Решение

Вж. UI/Solution.sln

Съдържание