**TASK 3 (energy accounting)**

**FILE PROGRAM.CS**

using System;

namespace hw3

{

class Program

{

static void Main(string[] args)

{

try

{

string path = @"energy.txt";

FileWorker fw = new FileWorker(path);

EnergyInfo energyInfo = new EnergyInfo(fw.GetNumberFlat(), fw.GetListOfCounters(), 2, (Quarter)fw.GetQuarter());

System.Console.WriteLine(energyInfo.ToString());

System.Console.WriteLine(energyInfo.ToString(23));

System.Console.WriteLine(energyInfo.FindBiggestBill());

System.Console.WriteLine(energyInfo.FindFlatWithoutUsing());

}

catch (Exception ex)

{

System.Console.WriteLine(ex.Message);

}

}

}

}

**FILE FILEWORKER.CS**

using System;

using System.IO;

namespace hw3

{

class FileWorker

{

string \_path;

public string Path

{

get => \_path; set

{

if (String.Compare(value, "") != 0)

{

\_path = value;

}

else

{

throw new ArgumentException("Path can't be empty");

}

}

}

public FileWorker(string path)

{

Path = path;

}

public int GetNumberFlat()

{

using (StreamReader file = new StreamReader(Path))

{

string data = file.ReadLine();

string[] info = data.Split(' ', StringSplitOptions.RemoveEmptyEntries);

int.TryParse(info[0], out int result);

if (result < 1)

{

throw new ArgumentException("Number of flats can't be negative number. Please, check your file");

}

return result;

}

}

public int GetQuarter()

{

using (StreamReader file = new StreamReader(Path))

{

string data = file.ReadLine();

string[] info = data.Split(' ', StringSplitOptions.RemoveEmptyEntries);

int.TryParse(info[1], out int result);

if (result < 1 || result > 4)

{

throw new ArgumentException("Quarter number isn't correct. Allowed values: 1, 2, 3, 4\nPlease, check your file");

}

return result;

}

}

public string[] GetListOfCounters()

{

using (StreamReader file = new StreamReader(Path))

{

string data = file.ReadLine();

string[] info = data.Split(' ', StringSplitOptions.RemoveEmptyEntries);

int.TryParse(info[0], out int flatNumber);

string[] result = new string[flatNumber];

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

{

result[i] = file.ReadLine();

}

return result;

}

}

}

}

**FILE ENERGYINFO.CS**

using System;

using System.Linq;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace hw3

{

public enum Quarter { First = 1, Second, Third, Fourth };

public enum Months

{

January = 1, February, March, April, May, June,

July, August, September, October, November, December

}

class EnergyInfo

{

private int \_flatNumber;

private List<FlatInfo> \_flatsInfo;

private double \_price;

private Quarter \_quarter;

public int FlatNumber

{

get => \_flatNumber;

set

{

if (value > 0)

{

\_flatNumber = value;

}

else

{

throw new ArgumentException("Number of flats must be > 0");

}

}

}

public List<FlatInfo> FlatsInfo { get => \_flatsInfo; }

public double Price

{

get => \_price;

set

{

if (value > 0)

{

\_price = value;

}

else

{

throw new ArgumentException("Price must be positive number");

}

}

}

public Quarter Quarter

{

get => \_quarter;

set

{

if (Enum.IsDefined(typeof(Quarter), value))

{

\_quarter = value;

}

else

{

throw new ArgumentException("This value isn't a quarter of a year");

}

}

}

public EnergyInfo(int flatCount, string[] flatsInfo, double price, Quarter quarter)

{

FlatNumber = flatCount;

\_flatsInfo = new List<FlatInfo>(flatCount);

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

{

string[] data = flatsInfo[i].Split(' ', StringSplitOptions.RemoveEmptyEntries);

int.TryParse(data[0], out int flatNumber);

string flatOwner = data[1];

(int, int)[] flatCounter = new (int, int)[3];

for (int j = 2, k = 0; j < data.Length - 1; j += 2, ++k)

{

int.TryParse(data[j], out flatCounter[k].Item1);

int.TryParse(data[j + 1], out flatCounter[k].Item2);

}

FlatsInfo.Add(new FlatInfo(flatNumber, flatOwner, flatCounter, price));

}

Price = price;

Quarter = quarter;

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

result.Append(String.Format("INFORMATION ABOUT ALL FLATS\n Total number of flats: {0}\n", FlatNumber));

result.Append(String.Format("Number of quarter: {0}\n", Quarter));

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

{

result.Append(FlatsInfo[i].ToString());

for (int j = 0; j < FlatsInfo[i].CounterInfo.Length; ++j)

{

result.Append(String.Format(" {0}:\n Begin: {1} kW\n End: {2} kW\n Total used: {3} kW\n",

(Months)(((int)Quarter - 1) \* 3 + j + 1),

FlatsInfo[i].CounterInfo[j].Item1,

FlatsInfo[i].CounterInfo[j].Item2,

FlatsInfo[i].CounterInfoTotal[j]));

}

result.Append(String.Format(" To pay: {0} UAH for {1}kW\n\n", FlatsInfo[i].Bill, FlatsInfo[i].CounterInfoTotal.Sum(item => item)));

}

return result.ToString();

}

public string FindBiggestBill()

{

StringBuilder result = new StringBuilder();

FlatInfo maxBillFlat = FlatsInfo.OrderByDescending(item => item.Bill).First();

result.Append(String.Format("Biggest bill:\n {0}", maxBillFlat));

return result.ToString();

}

public string FindFlatWithoutUsing()

{

StringBuilder result = new StringBuilder();

FlatInfo maxBillFlat = FlatsInfo.OrderBy(item => item.CounterInfoTotal.Sum(item => item)).First();

if (maxBillFlat.Bill != 0)

{

return "There is no flat without using electricity";

}

result.Append(String.Format("Flat without using electicity:\n {0}", maxBillFlat));

return result.ToString();

}

public string ToString(int flatNumber)

{

StringBuilder result = new StringBuilder();

foreach (FlatInfo item in FlatsInfo)

{

if (flatNumber == item.Number)

{

result.Append(String.Format("INFORMATION ABOUT SPECIFIC FLAT\nNumber of quarter: {0}\n", Quarter));

result.Append(item.ToString());

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

{

result.Append(String.Format(" {0}:\n Begin: {1} kW\n End: {2} kW\n Total used: {3} kW\n",

(Months)(((int)Quarter - 1) \* 3 + i + 1),

item.CounterInfo[i].Item1,

item.CounterInfo[i].Item2,

item.CounterInfoTotal[i]));

}

result.Append(String.Format(" To pay: {0} UAH for {1}kW\n\n", item.Bill, item.CounterInfoTotal.Sum(item => item)));

return result.ToString();

}

}

return "There is no flat with this number\n";

}

}

}

**FILE FLATINFO.CS**

using System;

using System.Linq;

using System.Text;

namespace hw3

{

class FlatInfo

{

private int \_number;

private string \_owner;

private (int, int)[] \_counterInfoBeginEnd;

private int[] \_countersInfoTotal;

private double \_price;

private double \_bill;

public int Number

{

get => \_number;

set

{

if (value > 0)

{

\_number = value;

}

else

{

throw new ArgumentException("Number of flats must be positive number");

}

}

}

public string Owner

{

get => \_owner;

set

{

if (String.Compare(value, "") != 0)

{

\_owner = value;

}

else

{

throw new ArgumentException("Owner's name can't be empty");

}

}

}

public (int, int)[] CounterInfo

{

get => \_counterInfoBeginEnd;

set

{

\_counterInfoBeginEnd = new (int, int)[value.Length];

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

{

\_counterInfoBeginEnd[i].Item1 = value[i].Item1;

\_counterInfoBeginEnd[i].Item2 = value[i].Item2;

}

}

}

public int[] CounterInfoTotal { get => \_countersInfoTotal; }

public double Price

{

get => \_price;

set

{

if (value > 0)

{

\_price = value;

}

else

{

throw new ArgumentException("Price must be a positive number");

}

}

}

public double Bill

{

get => \_bill;

private set

{

if (value >= 0)

{

\_bill = value;

}

else

{

throw new ArgumentException("Bill must be a positive number.\nCheck your price\\counters data");

}

}

}

public FlatInfo(int number, string owner, (int, int)[] counterData, double price)

{

Number = number;

Owner = owner;

CounterInfo = counterData;

Price = price;

\_countersInfoTotal = new int[counterData.Length];

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

{

\_countersInfoTotal[i] = counterData[i].Item2 - counterData[i].Item1;

}

Bill = \_countersInfoTotal.Sum(item => item \* Price);

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

result.Append(String.Format("Flat #{0} Owner: {1}\n", Number, Owner));

return result.ToString();

}

}

}

**TASK 4 (magical square)**

using System;

namespace hw3b

{

class Program

{

static void Main(string[] args)

{

try

{

MagicSquare matr = new MagicSquare(3);

System.Console.WriteLine(matr.ToString());

MagicSquare matr2 = new MagicSquare(5);

System.Console.WriteLine(matr2.IsSquare());

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

**FILE MAGICSQUARE.CS**

using System;

using System.Linq;

using System.Text;

namespace hw3b

{

class MagicSquare

{

private int[,] \_matrix;

private int \_matrixSize;

public int Size

{

get => \_matrixSize; set

{

if ((value > 0) && (value % 2 == 1))

{

\_matrixSize = value;

}

else

{

throw new ArgumentException("Size of matrix must be a positive odd number");

}

}

}

public int[,] Matrix

{

get => \_matrix;

}

public MagicSquare(int size)

{

Size = size;

\_matrix = new int[Size, Size];

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

{

for (int j = 0; j < Size; ++j)

{

\_matrix[i, j] = 0;

}

}

BuildSquare();

}

private void BuildSquare()

{

int i = Size / 2;

int j = Size - 1;

for (int num = 1; num <= Size \* Size;)

{

if (i == -1 && j == Size)

{

j = Size - 2;

i = 0;

}

else

{

if (j == Size)

{

j = 0;

}

if (i < 0)

{

i = Size - 1;

}

}

if (\_matrix[i, j] != 0)

{

j -= 2;

i++;

continue;

}

else

{

\_matrix[i, j] = num++;

}

j++;

i--;

}

IsSquare();

}

public bool IsSquare()

{

int[] matrixSums = new int[Size + Size + 2];

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

{

for (int j = 0; j < Size; ++j)

{

matrixSums[i] += \_matrix[i, j];

matrixSums[matrixSums.Length / 2 - 1 + i] += \_matrix[j, i];

}

}

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

{

matrixSums[matrixSums.Length - 2] += \_matrix[i, i];

matrixSums[matrixSums.Length - 1] += \_matrix[i, Size - i - 1];

}

bool result = matrixSums.Distinct().Count() == 1;

return result;

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

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

{

for (int j = 0; j < Size; ++j)

{

result.Append($"{Matrix[i, j],5} ");

}

result.Append("\n");

}

return result.ToString();

}

}

}