**ExamCardd**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ExamCardd

{

internal class **ExamCard**<T> : IAction<T> where T : new() //ограничение на конструктор

{

public static List<T> list = new List<T>();

void IAction<T>.Add( T a)

{

list.Add(a);

}

void IAction<T>.Delete( T a )

{

if (list.Count== 0)

{

throw new NullSizeCollection();

}

else

list.Remove(a);

}

void IAction<T>.Clear( )

{

if (list.Count == 0)

{

throw new NullSizeCollection();

}

list.Clear();

}

void IAction<T>.WatchColl()

{

Console.WriteLine("Коллекция: ");

foreach (var col in list)

{

Console.WriteLine(col);

}

}

}

}

internal interface **IAction<T>**

{

void Add(T a);

void Delete(T a);

void Clear();

void WatchColl();

}

internal class **NullSizeCollection**: System.Exception

{

public NullSizeCollection() {

Console.WriteLine("Пустая коллекция!");

}

// public NullSizeCollection(string message) : base(message){ }

}

internal class Student

{

public string Name;

public int Mark;

public string Subject;

public string name { get => Name; set=> Name= value; }

public int mark { get => Mark; set => Mark = value; }

public string subject { get => Subject; set => Subject = value; }

public Student() { }

public Student(string name, int mark,string subject)

{

Name = name;

Mark = mark;

Subject = subject;

}

public override string ToString()

{

return name + " " + mark +" " +subject;

}

}

static class Rasshir

{

public static void RandMark(this Student st)

{

Random random = new Random();

st.Mark += random.Next(1, 10);

}

}

class Program

{ static void Main( string[] args )

{

Student student1 = new Student("Senya",8,"OOP");

Student student2 = new Student("Vlad", 10, "OOP");

Student student3 = new Student("Liza", 7, "OOP");

Student student4 = new Student("Tani", 8, "OOP");

Student student5 = new Student("lexa", 3, "OOP");

Student student6 = new Student("Vika", 1, "OOP");

ExamCard<Student> studs = new ExamCard<Student>();

((IAction<Student>)studs).Add( student1);

((IAction<Student>)studs).Add(student2);

((IAction<Student>)studs).Add(student3);

((IAction<Student>)studs).Add(student4);

((IAction<Student>)studs).Add(student5);

((IAction<Student>)studs).Add(student6);

((IAction<Student>)studs).WatchColl();

// ((IAction<Student>)studs).Delete(student3);

// ((IAction<Student>)studs).Clear();

// ((IAction<Student>)studs).WatchColl();

//((IAction<Student>)studs).Delete( student1);

Console.WriteLine("------------------------------");

IEnumerable<Student> Exm = from t in ExamCard<Student>.list//public static List<T> list = new List<T>();

where t.Mark >=4

select t;

Console.WriteLine(Exm.Count());

foreach (Student i in Exm)

{

Console.WriteLine(i);

}

var ExmSr = from t in ExamCard<Student>.list

select t.Mark;

Console.WriteLine(ExmSr.Average());

student6.RandMark();

Console.WriteLine(student6)}}

**HUMAN PERSON BANK**

internal interface IScore

{

float Amount { get; set; }

void AddSum( float a );

void TakeSum( float a );

}

abstract class Human

{

DateOnly dateBirth {get; set;}

}

internal class Person :Human , IScore

{

public static int ObjKolvo;

public string Name;

private float amount;

public DateOnly DateBirth;

public float Amount

{

get=> amount;

set => amount = value;

}

public Person( string name, float amount, DateOnly dateBirth )

{

Name = name;

Amount = amount;

DateBirth = dateBirth;

ObjKolvo++;

}

public void AddSum(float a)

{

Amount += a;

}

public void TakeSum( float a )

{

Amount -= a;

}

public override string ToString()

{

return Name + " " + Amount + " " + DateBirth;

}

public override bool Equals( object? obj )

{

Person p = obj as Person;

if (p.DateBirth == DateBirth)

return true;

else

return false;

}

static Person()

{

ObjKolvo=0;

}

public static void KolOb()

{

Console.WriteLine($"Количесво объектов: {ObjKolvo}");

}

internal class Bank : List<Person>

{

public void Show( Bank b)

{

foreach (Person itm in b)

{

Console.WriteLine($"Имя {itm.Name} Сумма {itm.Amount} Дата рождения {itm.DateBirth}");

}}

}

static void Main( string[] args )

{

DateOnly date1 = new DateOnly(2002, 10, 27);

DateOnly date2 = new DateOnly(2000, 3, 06);

DateOnly date3 = new DateOnly(1995, 5, 15);

DateOnly date4 = new DateOnly(1995, 5, 15);

Person person = new Person("Sasha",15.41f,date1);

Person person2 = new Person("Roma", 75.21f, date2);

Person person3 = new Person("Vika", 119.02f, date3);

Person person4 = new Person("Jina", 73.11f, date4);

person.AddSum(10f);

Console.WriteLine(person);

person.TakeSum(10f);

Console.WriteLine(person);

Person.KolOb();

Console.WriteLine(person2.Equals(person3));

Console.WriteLine(person3.Equals(person4));

Console.WriteLine("------Bank------");

Bank belarus = new Bank();

belarus.Add(person);

belarus.Add(person2);

belarus.Add(person3);

Bank alfa = new Bank();

alfa.Add(person3);

alfa.Add(person4);

alfa.Add(person2);

Bank vtb = new Bank();

vtb.Add(person);

vtb.Add(person4);

vtb.Add(person3);

belarus.Show(belarus);

Console.WriteLine("------Bank------");

alfa.Show(alfa);

Console.WriteLine("------Bank------");

vtb.Show(vtb);

}

**STUDENT (3 notes) недодел**

public struct Stud

{

public string Name;

public int Group;

public int Course;

public Speciality Specialt;

public int Ex1;

public int Ex2;

public int Ex3;

public Stud( string name, int group,int course, Speciality specialt, int ex1, int ex2, int ex3 )

{

Name = name;

Group = group;

Course = course;

Specialt = specialt;

Ex1 = ex1;

Ex2 = ex2;

Ex3 = ex3;

}

public override string ToString()

{

return Name + " Group: " + Group + " Course: " + Course + " " + Specialt + " " + Ex1 + " " + Ex2 + " " + Ex3;

}

public Tuple<int, int, float> Notes ( )

{

int [] mas = { Ex1, Ex2, Ex3 };

int Max = mas.Max() ;

int Min = mas.Min();

float Avg= mas.Sum()/mas.Length;

return Tuple.Create(Max, Min, Avg);

}

public enum Speciality

{

poit,

isit,

mobile

}

class Program

{

static void Main( string[] args )

{

Stud stud1 = new Stud("Kit", 2, 2, Speciality.isit, 7, 4, 5);

Stud stud2 = new Stud("Sam", 7, 2, Speciality.poit, 5, 5, 7);

Stud stud3 = new Stud("Den", 4, 3, Speciality.mobile, 10, 10, 10);

Console.WriteLine(stud1);

Console.WriteLine(stud2);

Console.WriteLine(stud3);

Console.WriteLine("---------------");

Console.WriteLine("Max-Min-Avg");

Console.WriteLine(stud1.Notes());

}

// ------- 2. текущий домен с процессами ----------

**AppDomain domain = AppDomain.CurrentDomain;** // // возвращает текущий домен приложения для текущего объекта Thread, AppDomain - домен приложения, ктр является изолированной средой, в которой приложение

Console.WriteLine("\n\nТекущий домен: " + domain.FriendlyName);

Console.WriteLine("Базовый каталог: " + domain.BaseDirectory);

Console.WriteLine("Детали конфигурации: " + domain.SetupInformation);

Console.WriteLine("Все сборки в домене:\n");

foreach (Assembly ass in domain.GetAssemblies())

Console.WriteLine(ass.GetName().Name);

// ------- 3. записи в файл и на консоль простых чисел от 1 до n ----------

**Thread NumbersThread = new Thread(new ParameterizedThreadStart(WriteNums));** // создаем новый **поток**

Console.Write("\nЗадайте число: ");

int numb = Convert.ToInt32(Console.ReadLine());

NumbersThread.Start(numb); // запускаем его

Thread.Sleep(2000); // приостанавливает выполнение потока, в котором он был вызван

Console.WriteLine("--------------------");

Console.WriteLine("Приоритет: " + NumbersThread.Priority);

Thread.Sleep(100);

Console.WriteLine("Имя потока: " + NumbersThread.Name);

Thread.Sleep(100);

Console.WriteLine("ID потока: " + NumbersThread.ManagedThreadId);

Console.WriteLine("---------------------");

Thread.Sleep(1000);

Thread.Sleep(2000); // приостанавливает выполнение потока, в котором он был вызван

void WriteNums( object number )

{

int num = (int)number;

using (StreamWriter sw = new StreamWriter(@"F:\.Уник\2 курс\ООП С#\Лабы\lab14\lab14\numbers.txt", false, System.Text.Encoding.Default))

{

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

{

sw.WriteLine(i);

Console.WriteLine(i);

Thread.Sleep(500);

}

}

}

// System.Threading.Timer класс позволяет запускать определенные действия по истечению некоторого периода времени:

**TimerCallback timerCallback = new TimerCallback(WhatTimeIsIt);//** устанавливаем метод обратного вызова

Timer timer = new Timer(timerCallback, null, 500, 1000); /\* null - параметр, которого нет, 500 - время, через которое запустится процесс с таймером,

\* 1000 - периодичность таймера (интервал между вызовами метода делегата). \*/

Thread.Sleep(5000);

// 500 - ждем и не закрываем поток

timer.Change(Timeout.Infinite, 2000); // уничтожение таймера

void WhatTimeIsIt( object obj )

{

Console.WriteLine($"It's {DateTime.Now.Hour}:{DateTime.Now.Minute}:{DateTime.Now.Second}");

}

for (int i = 0; i < 5; i++)

{

**Stopwatch sw = new Stopwatch();** // точное измерение затраченного времени

sw.Start();

Task task = new Task(() => MulByVector(10000)); //умножение вектора размера 100000 на число,

**task.Start();**

Console.WriteLine($"ID: {task.Id}, статус: {task.Status}");

task.Wait(); //останавливают основной поток до завершения задачи

Console.WriteLine($"ID: {task.Id}, статус: {task.Status}");

sw.Stop();

Console.WriteLine($"Производительность: {sw.ElapsedMilliseconds}ms");//св-во Получает общее затраченное время в миллисекундах, измеренное текущим экземпляром.

Console.WriteLine();

}

}

static void MulByVector( int k, object ob = null )

{

Random random = new Random();

List<int> vector = new List<int>();

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

{

vector.Add(random.Next(1, 10));

}

vector = vector.Select(x => x \* 10).ToList();

}

**Task<int> first = new Task<int>**(() => new Random().Next(1, 9) \* 100);

Task<int> second = new Task<int>(() => new Random().Next(0, 9) \* 10);

Task<int> third = new Task<int>(() => new Random().Next(0, 9));

first.Start();

second.Start();

third.Start();

first.Wait();

second.Wait();

third.Wait();

Task<int> number = new Task<int>(() => first.Result + second.Result + third.Result);//описывает задачу, возвращающую значение типа < >

number.Start();

Console.WriteLine($"3. Number: {number.Result}");

public static CollectionType<T> **operator +(** CollectionType<T> oper, T item ) //перегрузка операторов{

Console.WriteLine("Перегрузка оператора \"+\"");

oper.Push(item);

oper.kolvo++;

return oper;

}

public static CollectionType<T> **operator --**( CollectionType<T> oper )

{

Console.WriteLine("Перегрузка оператора \"-\"");

oper.Pop();

oper.kolvo--;

return oper;

}

public static bool **operator true**( CollectionType<T> v )

{

if (v.Count() != 0)

return true;

else

return false;

}

//запрос возвращающий только летние и зимние месяцы

**IEnumerable**<string> WintSummer = from m in Month

where m == "June" || m == "July" || m == "August"|| m == "January" || m == "December" || m == "February"

select m;

Console.WriteLine("\nлетние и зимние месяцы");

foreach (string i in WintSummer)

{

Console.WriteLine(i);

}

//запрос вывода месяцев в алфавитном порядке

IEnumerable<string> AlfMonth = from m in Month

orderby m ascending

select m;

Console.WriteLine("\nВ алфавитном порядке");

foreach (string i in AlfMonth)

{

Console.WriteLine(i);

}

var MaxDay = Airlist

.GroupBy(n => n.Weekday)

.Select(n => new { День\_недели = n.Key, Максимальный\_номер\_рейса = n.Max(x => x.Flight\_Num) });

Console.WriteLine("\n------максимальны по дню недели рейс------");

foreach (var i in MaxDay)

{

Console.WriteLine(i);

}

namespace lab8{

public **delegate** void RenameEventHandler( string name );

public delegate void NewPropertyEventHandler( string name );

class{

public **event** RenameEventHandler Rename;

public event NewPropertyEventHandler NewProperty;

public void CommandNewProp( string n )

{NewProperty.Invoke(n); // вызывает метод в том потоке, где он был создан

Console.WriteLine("\n-- Вызвано событие изменения свойств --");

}

public void CommandRenameOperation( string n ) {

Rename.Invoke(n);

Console.WriteLine("\n-- Вызвано событие Rename --");

}