# <https://ulearn.me/Course/cs2/Uprazhnenie_na_parametry_po_umolchaniyu_d6165afa-81b2-452f-9e61-78d3580e0c27>

Отрефакторьте класс MyFile, чтобы в нем остался только один метод.

# Решение

class MyFile

{

public static int GetMethodsCount(){

return 1;

}

 public static string ReadAll(string filename, Encoding enc)

 {

 Console.WriteLine("Use encoding " + enc);

 return null;

 }

 public static string ReadAll(string filename)

 {

 return ReadAll(filename, Encoding.UTF8);

 }

}

# <https://ulearn.me/Course/cs2/Uprazhnenie_na_operatory_9ea26732-f12b-4d2d-b1f2-9e672e2a1f22>

Напишите как можно меньше кода, чтобы программа скомпилировалась.

# Решение

class A{

 int value = 1;

 public static int operator +(A p1, A p2)

 {

 return new A { value = p1.value + p2.value };

 }

 public static int operator \*(A p1, string p2)

 {

 return new A { value = p1.value \* p2.Length };

 }

 public static implicit operator int(A p)

 {

 return p.value;

 }

}

# <https://ulearn.me/Course/cs2/Neozhidannyy_Hello_world__64a9aaab-35ff-4b81-ac23-82484e782e8c>

Сделайте так, чтобы эта программа вывела на консоль Hello, world!

# Решение

class A

{

 private int number;

 public int Number

 {

 get { return number; }

 set {

 Console.WriteLine("Hello, world!");

 number = value;

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Sboi__232d980f-d135-4f22-bf7b-ce39b66ce111>

Некто N. написал код, выводящий список устройств, в которых за последний месяц до определенной даты случились критические сбои. К сожалению, N. учился программированию в начале 90-х годов, и не знаком с современными практиками.

Скачайте [проект Incapsulation.Failures](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=232d980f-d135-4f22-bf7b-ce39b66ce111) и помогите N. отрефакторить его код:

1. Выделите новый статический метод FindDevicesFailedBeforeDate. Метод должен принимать не более 4-х аргументов. В сигнатуре метода не должно быть Dictionary-типов и коллекций с вложенными дженерик-типами, например, List<List<object>>.
2. Значения в аргументах devices и failureTypes должны быть инкапсулированы в сущности Device и Failure.
3. IsFailureSerious, очевидно, не на своем месте.
4. С day и times тоже не все в порядке.

Сигнатуру старого метода сохраните, чтобы проходили тесты. Старый метод должен преобразовывать аргументы и вызывать новый метод.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Incapsulation.Failures

{

 public class Common

 {

 public static bool IsFailureSerious(FailureType failureType)

 {

 if (failureType == FailureType.UnexpectedShutdown || failureType == FailureType.HardwareFailures)

 {

 return true;

 }

 return false;

 }

 public static bool Earlier(DateTime v, DateTime d)

 {

 return v < d;

 }

 public static FailureType GetEnumByIndex(int i)

 {

 if (i == 0)

 {

 return FailureType.UnexpectedShutdown;

 }

 if (i == 1)

 {

 return FailureType.ShortNonResponding;

 }

 if (i == 2)

 {

 return FailureType.HardwareFailures;

 }

 return FailureType.ConnectionProblems;

 }

 public static Device GetDeviceByID(Device[] d, int id)

 {

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

 {

 if (d[i].ID == id)

 {

 return d[i];

 }

 }

 return null;

 }

 }

 public class Device

 {

 public int ID;

 public string Name;

 public Device(int pID, string pName)

 {

 ID = pID;

 Name = pName;

 }

 }

 public enum FailureType

 {

 UnexpectedShutdown,

 ShortNonResponding,

 HardwareFailures,

 ConnectionProblems

 }

 public class ReportMaker

 {

 public static List<string> FindDevicesFailedBeforeDate(

 DateTime date,

 FailureType[] failureTypes,

 DateTime[] times,

 Device[] devices)

 {

 var problematicDevices = new HashSet<int>();

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

 {

 if (Common.IsFailureSerious(failureTypes[i]) == true && Common.Earlier(times[i], date) == true) {

 problematicDevices.Add(devices[i].ID);

 }

 }

 var result = new List<string>();

 foreach (var device in devices)

 if (problematicDevices.Contains(device.ID)) {

 result.Add(device.Name);

 }

 return result;

 }

 public static List<string> FindDevicesFailedBeforeDateObsolete(

 int day,

 int month,

 int year,

 int[] failureTypes,

 int[] deviceId,

 object[][] times,

 List<Dictionary<string, object>> devices)

 {

 DateTime d = new DateTime(year, month, day);

 FailureType[] f = new FailureType[failureTypes.Length];

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

 f[i] = Common.GetEnumByIndex(failureTypes[i]);

 }

 DateTime[] t = new DateTime[times.Length];

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

 {

 t[i] = new DateTime((int)times[i][2], (int)times[i][1], (int)times[i][0]);

 }

 Device[] dev = new Device[devices.Count];

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

 dev[i] = new Device((int)devices[i]["DeviceId"], (string)devices[i]["Name"]);

 }

 Device[] dev\_final = new Device[devices.Count];

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

 {

 dev\_final[i] = Common.GetDeviceByID(dev, deviceId[i]);

 }

 return FindDevicesFailedBeforeDate(d,f,t, dev\_final);

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Predpriyatie__45750d32-5263-4d20-b533-c1ed3d809dd6>

Некто M. написал код, описывающий предприятие. Он даже озаботился проверкой целостности для полей этого класса, но, к сожалению, он учился программировать в конце 90-х годов, и знаком лишь со слегка устаревшими практиками проверки целостности.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Incapsulation.EnterpriseTask

{

 public class Enterprise

 {

 private readonly Guid guid;

 public Guid Guid { get; }

 public Enterprise(Guid pGuid)

 {

 guid = pGuid;

 }

 public string Name { get; set; }

 private string inn;

 public string Inn {

 get

 {

 return inn;

 }

 set

 {

 if (inn.Length != 10 || !inn.All(z => char.IsDigit(z)))

 throw new ArgumentException();

 inn = value;

 }

 }

 public DateTime EstablishDate { get; set; }

 public TimeSpan ActiveTimeSpan

 {

 get

 {

 return DateTime.Now - EstablishDate;

 }

 }

 public double GetTotalTransactionsAmount()

 {

 DataBase.OpenConnection();

 var amount = 0.0;

 foreach (Transaction t in DataBase.Transactions().Where(z => z.EnterpriseGuid == guid))

 amount += t.Amount;

 return amount;

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Vesa__1589a548-0f62-40ba-949a-985b6424cc64>

Скачайте [проект Incapsulation.Weights](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=1589a548-0f62-40ba-949a-985b6424cc64) и напишите класс Indexer, который создается как обертка над массивом double[], и открывает доступ к его подмассиву некоторой длины, начиная с некоторого элемента. Ваше решение должно проходить тесты, содержащиеся в проекте. Как и всегда, вы должны следить за целостностью данных в Indexer.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Incapsulation.Weights

{

 class Indexer

 {

 private static double[] SubArray;

 private static bool Init = false;

 private int \_Start;

 private int \_Length;

 public int Length

 {

 get

 {

 return \_Length;

 }

 }

 public int Start

 {

 get

 {

 return \_Start;

 }

 }

 public double this[int index]

 {

 get

 {

 if (index < 0)

 {

 throw new IndexOutOfRangeException("Индекс не может быть меньше 0.");

 }

 if (index > Length-1)

 {

 throw new IndexOutOfRangeException("Индекс не может быть больше длины вектора.");

 }

 return SubArray[Start + index];

 }

 set

 {

 if (index < 0)

 {

 throw new IndexOutOfRangeException("Индекс не может быть меньше 0.");

 }

 if (index > Length-1)

 {

 throw new IndexOutOfRangeException("Индекс не может быть больше длины вектора.");

 }

 SubArray[Start + index] = value;

 }

 }

 public Indexer(double[] ar, int start, int length)

 {

 if (start < 0)

 {

 throw new ArgumentException("Индекс начала вектора не может быть отрицательным.");

 }

 if (length < 0)

 {

 throw new ArgumentException("Длина вектора не может быть отрицательной.");

 }

 if (length > ar.Length)

 {

 throw new ArgumentException("Длина вектора не может быть больше, чем длина исходного массива.");

 }

 if (length > ar.Length - start)

 {

 throw new ArgumentException("Длина вектора не может быть больше, чем длина исходного массива.");

 }

 \_Length = length;

 \_Start = start;

 if (Init == false)

 {

 SubArray = ar;

 }

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Ratsional_nye_chisla__bd5ee0e1-e433-4624-8b6c-96a6b7a81125>

При работе над математическими или геометрическими задачами часто приходится создавать "фундаментальные" классы, подобные int или double, для комплексных чисел, кватернионов и т.д. Скачайте [проект Incapsulation.RationalNumbers](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=bd5ee0e1-e433-4624-8b6c-96a6b7a81125) и напишите класс рационального числа.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Incapsulation.RationalNumbers

{

 public class Rational

 {

 private int \_numerator;

 private int \_denominator;

 // Публичные свойства и методы

 public int Numerator

 {

 get

 {

 return \_numerator;

 }

 }

 public int Denominator

 {

 get

 {

 return \_denominator;

 }

 }

 public bool IsNan

 {

 get

 {

 bool cond = false;

 if (\_denominator == 0)

 {

 cond = true;

 }

 return cond;

 }

 }

 // Приватные методы

 private int Nod()

 {

 int n = Math.Abs(\_numerator);

 int d = Math.Abs(\_denominator);

 while (d != 0 && n != 0)

 {

 if (n % d > 0)

 {

 var temp = n;

 n = d;

 d = temp % d;

 }

 else break;

 }

 if (d != 0 && n != 0) return d;

 return 1;

 }

 private void Reduce()

 {

 int nod = Nod();

 \_numerator /= nod;

 \_denominator /= nod;

 }

 private static void CommonDenominator(Rational a, Rational b)

 {

 if (a.\_denominator != b.\_denominator)

 {

 int common = a.\_denominator \* b.\_denominator;

 int mn\_a = common / a.\_denominator;

 int mn\_b = common / b.\_denominator;

 a.\_denominator = common;

 a.\_numerator = a.\_numerator \* mn\_a;

 b.\_denominator = common;

 b.\_numerator = b.\_numerator \* mn\_b;

 }

 }

 // Конструкторы

 public Rational(int a, int b)

 {

 if (b < 0)

 {

 a \*= -1;

 }

 \_numerator = a;

 \_denominator = Math.Abs(b);

 Reduce();

 }

 public Rational(int a)

 {

 \_numerator = a;

 \_denominator = 1;

 }

 // Операторы

 public static Rational operator +(Rational a, Rational b)

 {

 if (a.Denominator == 0 || b.Denominator == 0)

 {

 return new Rational(0, 0);

 }

 else

 {

 CommonDenominator(a, b);

 return new Rational(a.Numerator + b.Numerator, a.Denominator);

 }

 }

 public static Rational operator -(Rational a, Rational b)

 {

 if (a.Denominator == 0 || b.Denominator == 0)

 {

 return new Rational(0, 0);

 }

 else {

 CommonDenominator(a, b);

 return new Rational(a.Numerator - b.Numerator, a.Denominator);

 }

 }

 public static Rational operator \*(Rational a, Rational b)

 {

 return new Rational(a.Numerator \* b.Numerator, a.Denominator \* b.Denominator);

 }

 public static Rational operator /(Rational a, Rational b)

 {

 if (a.Denominator == 0 || b.Denominator == 0)

 {

 return new Rational(a.Numerator \* b.Denominator, 0);

 } else

 {

 return new Rational(a.Numerator \* b.Denominator, a.Denominator \* b.Numerator);

 }

 }

 // Приведение типов

 public static implicit operator double(Rational a)

 {

 return (double)a.Numerator / (double)a.Denominator;

 }

 public static implicit operator Rational(int a)

 {

 return new Rational(a);

 }

 public static explicit operator int(Rational a)

 {

 double res = (double)a.Numerator / (double)a.Denominator;

 double dec = Math.Truncate(res);

 if (res - dec == 0)

 {

 return (int)dec;

 } else

 {

 throw new ArgumentException("Не удается преобразовать дробь в целое число");

 }

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Struktura_dannykh__1cb3fef1-004f-45df-a94a-412aa744ed2a>

Скачайте [проект Inheritance.DataStucture](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=1cb3fef1-004f-45df-a94a-412aa744ed2a) и создайте класс Category.cs. В этом классе переопределите методы Equals и GetHashCode, реализуйте интерфейс IComparable, упорядочивающий категории сначала по продукту, затем по типу и затем - по теме, а также реализуйте все операторы сравнения. Изучите тесты для того, чтобы понять детали реализации.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Inheritance.DataStructure

{

 public class Category : IComparable

 {

 private string \_message;

 private MessageType \_type;

 private MessageTopic \_topic;

 public string Message

 {

 get

 {

 return \_message;

 }

 }

 public MessageType Type

 {

 get

 {

 return \_type;

 }

 }

 public MessageTopic Topic

 {

 get

 {

 return \_topic;

 }

 }

 public int CompareTo(object o)

 {

 try

 {

 Category c = o as Category;

 // Сначала сравниваем по Message

 if (String.Compare(Message, c.Message) < 0)

 {

 return -1;

 }

 else

 {

 if (String.Compare(Message, c.Message) == 0)

 {

 // Потом сравниваем по типу сообщения

 if (Type < c.Type)

 {

 return -1;

 }

 else

 {

 if (Type == c.Type)

 {

 // В конце сравниваем по теме сообщения

 if (Topic < c.Topic)

 {

 return -1;

 }

 else

 {

 if (Topic == c.Topic)

 {

 return 0;

 }

 else

 {

 return 1;

 }

 }

 }

 else

 {

 return 1;

 }

 }

 }

 else

 {

 return 1;

 }

 }

 } catch(NullReferenceException)

 {

 return -1;

 }

 }

 public override bool Equals(object o)

 {

 try

 {

 Category c = o as Category;

 return (this == c);

 } catch (NullReferenceException)

 {

 return false;

 }

 }

 public override string ToString()

 {

 return String.Format("{0}.{1}.{2}",Message,Type,Topic);

 }

 public Category(string pMessage, MessageType pType, MessageTopic pTopic)

 {

 \_message = pMessage;

 \_type = pType;

 \_topic = pTopic;

 }

 // Операторы

 public static bool operator >(Category a, Category b)

 {

 return a.CompareTo(b) == 1;

 }

 public static bool operator <(Category a, Category b)

 {

 return a.CompareTo(b) == -1;

 }

 public static bool operator >=(Category a, Category b)

 {

 return (a.CompareTo(b) == 1 || a.CompareTo(b) == 0);

 }

 public static bool operator <=(Category a, Category b)

 {

 return (a.CompareTo(b) == -1 || a.CompareTo(b) == 0);

 }

 public static bool operator ==(Category a, Category b)

 {

 return a.CompareTo(b) == 0;

 }

 public static bool operator !=(Category a, Category b)

 {

 return a.CompareTo(b) != 0;

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_HoMM__e699b946-6ed6-4aee-8618-35eb917537d9>

Выделите все поля, необходимую для каждого взаимодействия, в свой интерфейс. Отрефакторьте программу, избавившись от повторяющихся участков кода в Interaction.Make.

В итоговом решении Interaction.Make должен работать только с интерфейсами, и не должен содержать упоминаний конкретных классов

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Inheritance.MapObjects

{

 public interface IMapObject

 {

 void Interact(Player p);

 }

 public interface IArmy

 {

 Army Army { get; set; }

 }

 public interface IOwner

 {

 int Owner { get; set; }

 }

 public interface ITreasure

 {

 Treasure Treasure { get; set; }

 }

 public class Dwelling : IMapObject

 {

 public int Owner { get; set; }

 public void Interact(Player p)

 {

 Owner = p.Id;

 }

 }

 public class Mine : IMapObject, IArmy, IOwner

 {

 public int Owner { get; set; }

 public Army Army { get; set; }

 public Treasure Treasure { get; set; }

 public void Interact(Player p)

 {

 if (p.CanBeat(Army))

 {

 Owner = p.Id;

 p.Consume(Treasure);

 }

 else p.Die();

 }

 }

 public class Creeps : IMapObject, IArmy

 {

 public Army Army { get; set; }

 public Treasure Treasure { get; set; }

 public void Interact(Player p)

 {

 if (p.CanBeat(Army))

 {

 p.Consume(Treasure);

 }

 else {

 p.Die();

 }

 }

 }

 public class Wolfs : IMapObject

 {

 public Army Army { get; set; }

 public void Interact(Player p)

 {

 if (!p.CanBeat(Army)) {

 p.Die();

 }

 }

 }

 public class ResourcePile : IMapObject

 {

 public Treasure Treasure { get; set; }

 public void Interact(Player p)

 {

 p.Consume(Treasure);

 }

 }

 public static class Interaction

 {

 public static void Make(Player player, IMapObject obj)

 {

 obj.Interact(player);

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Geometriya_1__c3883d8d-76a7-4139-9713-b8e4a812cd3b>

Предположим вы знаете, что в планах добавить ещё много новых геометрических примитивов. В этом случае разумно сделать метод GetVolume абстрактным и переопределить его в классах Cube, Ball и Cyllinder.

Сделайте это!

В финальном решении не должно быть ни одного приведения типа.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Inheritance.Geometry

{

 public abstract class Body

 {

 public abstract double GetVolume();

 }

 public class Ball : Body

 {

 public double Radius { get; set; }

 public override double GetVolume()

 {

 return 4.0 \* Math.PI \* Math.Pow(Radius, 3) / 3;

 }

 }

 public class Cube : Body

 {

 public double Size { get; set; }

 public override double GetVolume()

 {

 return Math.Pow(Size, 3);

 }

 }

 public class Cyllinder : Body

 {

 public double Height { get; set; }

 public double Radius { get; set; }

 public override double GetVolume()

 {

 return Math.PI \* Math.Pow(Radius, 2) \* Height;

 }

 }

 // Заготовка класса для задачи на Visitor

 public class SurfaceAreaVisitor

 {

 public double SurfaceArea { get; private set; }

 }

 public class DimensionsVisitor

 {

 public Dimensions Dimensions { get; private set; }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Geometriya_2__69814431-f1ac-48ba-b796-739cdffa262c>

Определите интерфейс IVisitor и реализуйте его в двух классах DimensionsVisitor и SurfaceAreaVisitor, для рассчёта размеров (ширина, высота) и площади поверхности фигур.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Inheritance.Geometry

{

 public interface IVisitor

 {

 void Visit(Ball ball);

 void Visit(Cube cube);

 void Visit(Cyllinder cyllinder);

 }

 public abstract class Body

 {

 public abstract double GetVolume();

 public abstract void Accept(IVisitor visitor);

 }

 public class Ball : Body

 {

 public double Radius { get; set; }

 public override double GetVolume()

 {

 return 4.0 \* Math.PI \* Math.Pow(Radius, 3) / 3;

 }

 public override void Accept(IVisitor visitor)

 {

 visitor.Visit(this);

 }

 }

 public class Cube : Body

 {

 public double Size { get; set; }

 public override double GetVolume()

 {

 return Math.Pow(Size, 3);

 }

 public override void Accept(IVisitor visitor)

 {

 visitor.Visit(this);

 }

 }

 public class Cyllinder : Body

 {

 public double Height { get; set; }

 public double Radius { get; set; }

 public override double GetVolume()

 {

 return Math.PI \* Math.Pow(Radius, 2) \* Height;

 }

 public override void Accept(IVisitor visitor)

 {

 visitor.Visit(this);

 }

 }

 // Заготовка класса для задачи на Visitor

 public class SurfaceAreaVisitor : IVisitor

 {

 public double SurfaceArea { get; private set; }

 public void Visit(Ball ball)

 {

 SurfaceArea = 4 \* Math.PI \* ball.Radius \* ball.Radius;

 }

 public void Visit(Cube cube)

 {

 SurfaceArea = 6 \* cube.Size \* cube.Size;

 }

 public void Visit(Cyllinder cyllinder)

 {

 SurfaceArea = 2 \* Math.PI \* cyllinder.Radius \* (cyllinder.Height + cyllinder.Radius);

 }

 }

 public class DimensionsVisitor : IVisitor

 {

 public Dimensions Dimensions { get; private set; }

 public void Visit(Ball ball)

 {

 Dimensions = new Dimensions(ball.Radius \* 2, ball.Radius \* 2);

 }

 public void Visit(Cube cube)

 {

 Dimensions = new Dimensions(cube.Size, cube.Size);

 }

 public void Visit(Cyllinder cyllinder)

 {

 Dimensions = new Dimensions(cyllinder.Radius \* 2, cyllinder.Height);

 }

 }

}

<https://ulearn.me/Course/cs2/Praktika_Binarnoe_derevo__f9d7fc6d-4dd6-40ee-8ce2-bc174b0fae74>

Наиболее очевидный случай использования дженериков — создание коллекций. Скачайте [проект Generics.BinaryTrees](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=f9d7fc6d-4dd6-40ee-8ce2-bc174b0fae74) и создайте в нем класс [бинарного дерева поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0) так, чтобы он проходил приложенные тесты.

Если у вас останется много времени, оптимизируйте код метода GetEnumerable так, чтобы он работал за O(n) по времени, где n — количество элементов в дереве, и O(1) по памяти. Если времени не останется, вы можете использовать менее оптимальное решение.

# Решение

<https://ulearn.me/Course/cs2/Praktika_Tablitsy__4fdc50f8-ca3a-4f90-959d-e2c1de50fe69>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Generics.Tables

{

 public class Cell<TypeRows, TypeColumns, TypeCells> where TypeRows : IComparable

 where TypeColumns : IComparable

 where TypeCells : IComparable

 {

 public TypeRows Row;

 public TypeColumns Column;

 public TypeCells Data;

 public Cell(TypeRows pRow, TypeColumns pColumn, TypeCells pData)

 {

 Row = pRow;

 Column = pColumn;

 Data = pData;

 }

 }

 public class CRows<TypeRows> where TypeRows : IComparable

 {

 public List<TypeRows> Indexes = new List<TypeRows>();

 public int Count()

 {

 return Indexes.Count();

 }

 public bool HasIndex(TypeRows p)

 {

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

 {

 if (Indexes[i].CompareTo(Indexes[i]) == 0)

 {

 return true;

 }

 }

 return false;

 }

 }

 public class CColumns<TypeColumns> where TypeColumns : IComparable

 {

 public List<TypeColumns> Indexes = new List<TypeColumns>();

 public int Count()

 {

 return Indexes.Count();

 }

 public bool HasIndex(TypeColumns p)

 {

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

 {

 if (Indexes[i].CompareTo(Indexes[i]) == 0)

 {

 return true;

 }

 }

 return false;

 }

 }

 public class COpen<TypeRows, TypeColumns, TypeCells> where TypeRows : IComparable

 where TypeColumns : IComparable

 where TypeCells : IComparable

 {

 public Table<TypeRows, TypeColumns, TypeCells> Reference;

 public TypeCells this[TypeRows index\_row, TypeColumns index\_column]

 {

 set

 {

 int index\_global = Reference.GetCellIndex(index\_row, index\_column);

 if (index\_global == -1)

 {

 Reference.Cells.Add(new Cell<TypeRows, TypeColumns, TypeCells>(index\_row, index\_column, value));

 if (Reference.Rows.HasIndex(index\_row) == false)

 {

 Reference.Rows.Indexes.Add(index\_row);

 }

 if (Reference.Columns.HasIndex(index\_column) == false)

 {

 Reference.Columns.Indexes.Add(index\_column);

 }

 } else

 {

 Reference.Cells[index\_global].Data = value;

 }

 }

 get

 {

 int index\_global = Reference.GetCellIndex(index\_row, index\_column);

 if (Reference.Columns.HasIndex(index\_column) && Reference.Rows.HasIndex(index\_row))

 {

 if (index\_global != -1)

 {

 return Reference.Cells[index\_global].Data;

 }

 }

 return (TypeCells)Convert.ChangeType("0", typeof(TypeCells));

 }

 }

 }

 enum ZeroEnum

 {

 Value = 0

 }

 public class CExisted<TypeRows, TypeColumns, TypeCells> where TypeRows : IComparable

 where TypeColumns : IComparable

 where TypeCells : IComparable

 {

 public Table<TypeRows, TypeColumns, TypeCells> Reference;

 public TypeCells this[TypeRows index\_row, TypeColumns index\_column]

 {

 set

 {

 int index\_global = Reference.GetCellIndex(index\_row, index\_column);

 if (Reference.Columns.HasIndex(index\_column) && Reference.Rows.HasIndex(index\_row))

 {

 if (index\_global == -1)

 {

 Reference.Cells.Add(new Cell<TypeRows, TypeColumns, TypeCells>(index\_row, index\_column, value));

 } else

 {

 Reference.Cells[index\_global].Data = value;

 }

 } else

 {

 throw new ArgumentException(String.Format("Вы пытаетесь установить значение для несуществующего элемента с адресом [{0}][{1}]", index\_row, index\_column));

 }

 }

 get

 {

 int index\_global = Reference.GetCellIndex(index\_row, index\_column);

 if (Reference.Columns.HasIndex(index\_column) && Reference.Rows.HasIndex(index\_row))

 {

 if (index\_global == -1)

 {

 return (TypeCells)Convert.ChangeType("0", typeof(TypeCells));

 } else

 {

 return Reference.Cells[index\_global].Data;

 }

 } else

 {

 throw new ArgumentException(String.Format("Нет элемента с адресом [{0}][{1}]", index\_row, index\_column));

 }

 }

 }

 }

 public class Table<TypeRows, TypeColumns, TypeCells> where TypeRows : IComparable

 where TypeColumns : IComparable

 where TypeCells : IComparable

 {

 public CRows<TypeRows> Rows;

 public CColumns<TypeColumns> Columns;

 public List<Cell<TypeRows, TypeColumns, TypeCells>> Cells;

 public COpen<TypeRows, TypeColumns, TypeCells> Open;

 public CExisted<TypeRows, TypeColumns, TypeCells> Existed;

 public void AddRow(TypeRows index\_row)

 {

 if (Rows.HasIndex(index\_row) == false)

 {

 Rows.Indexes.Add(index\_row);

 }

 }

 public void AddColumn(TypeColumns index\_column)

 {

 if (Columns.HasIndex(index\_column) == false)

 {

 Columns.Indexes.Add(index\_column);

 }

 }

 public int GetCellIndex(TypeRows index\_row, TypeColumns index\_column)

 {

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

 {

 if (Cells[i].Row.CompareTo(index\_row) == 0 && Cells[i].Column.CompareTo(index\_column) == 0)

 {

 return i;

 }

 }

 return -1;

 }

 public Table()

 {

 Rows = new CRows<TypeRows>();

 Columns = new CColumns<TypeColumns>();

 Open = new COpen<TypeRows, TypeColumns, TypeCells>();

 Open.Reference = this;

 Existed = new CExisted<TypeRows, TypeColumns, TypeCells>();

 Existed.Reference = this;

 Cells = new List<Cell<TypeRows, TypeColumns, TypeCells>>();

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Roboty__58ad2ca7-bb8b-4334-9921-3ba18e8da67e>

Скачайте [проект Generics.Robots](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=58ad2ca7-bb8b-4334-9921-3ba18e8da67e) и изучите класс Architecture.cs. Он описывает некий проект архитектуры робота. В нем есть AI, вырабатывающий команды, и Device, команды исполняющий.

При этом, AI уже готовы для двух типов роботов (Builder и Shooter), а Device есть только для подвижной части.

Сейчас все работает, но вам не нравится. Что это за object-ы повсюду, где строгая типизация? Конечно, RobotAI и Mover должны стать дженерик-классами, типизируемыми классом команды. Однако, когда вы это сделаете, вы обнаружите, что эта архитектура не компилируется. Здесь нужно применить ковариацию для того, чтобы исправить эту проблему

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace Generics.Robots

{

 public interface RobotAI<out IMoveCommand>

 {

 IMoveCommand GetCommand();

 }

 public class ShooterAI : RobotAI<ShooterCommand>

 {

 int counter = 1;

 public ShooterCommand GetCommand()

 {

 return ShooterCommand.ForCounter(counter++);

 }

 }

 public class BuilderAI : RobotAI<BuilderCommand>

 {

 int counter = 1;

 public BuilderCommand GetCommand()

 {

 return BuilderCommand.ForCounter(counter++);

 }

 }

 public interface Device<in IMoveCommand>

 {

 string ExecuteCommand(IMoveCommand command);

 }

 public class Mover : Device<IMoveCommand>

 {

 public virtual string ExecuteCommand(IMoveCommand \_command)

 {

 IMoveCommand command = (IMoveCommand)\_command;

 if (command == null)

 throw new ArgumentException();

 return $"MOV {command.Destination.X}, {command.Destination.Y}";

 }

 }

 public class Robot

 {

 RobotAI<IMoveCommand> ai;

 Device<IMoveCommand> device;

 public Robot(RobotAI<IMoveCommand> ai, Device<IMoveCommand> executor)

 {

 this.ai = ai;

 this.device = executor;

 }

 public IEnumerable<string> Start(int steps)

 {

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

 {

 var command = ai.GetCommand();

 if (command == null)

 break;

 yield return device.ExecuteCommand(command);

 }

 }

 public static Robot Create(RobotAI<IMoveCommand> ai, Device<IMoveCommand> executor)

 {

 return new Robot(ai, executor);

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Generator_otchetov__d2604783-30f2-4fd9-be2e-5965c866857e>

Некто N разработал [генератор отчетов в проекте Delegates.Reports](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=d2604783-30f2-4fd9-be2e-5965c866857e), который считает простую статистику о погоде по нескольким параметрам за несколько дней. Его генератор расширяем, и он написал два отчета с его помощью: отчет в HTML, считающий среднее и стандартное отклонение, и отчет в Markdown, считающий медианы.

Однако, что делать, если нужно посчитать медианы и вывести результат в HTML? Что, если нужен будет третий отчет в HTML? Текущее решение крайне неудобно для таких ситуаций.

Помогите N отрефакторить код, переведя его с наследования на делегирование. Разделите ответственности по оформлению отчета и по вычислению показателей. В результате сам класс ReportMaker вам, возможно, уже и не понадобится.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Globalization;

namespace Delegates.Reports

{

 public delegate string ReportDelegate(ReportMaker printer, List<Measurement> pData);

 public abstract class ReportMaker

 {

 public abstract string MakeCaption(string caption);

 public abstract string BeginList();

 public abstract string MakeItem(string valueType, string entry);

 public abstract string EndList();

 public abstract string Caption { get; }

 public string MakeReport(ReportDelegate ReportMethod, ReportMaker printer, List<Measurement> pData)

 {

 return ReportMethod(printer, pData);

 }

 /\*

 {

 var data = measurements.ToList();

 var result = new StringBuilder();

 result.Append(MakeCaption(Caption));

 result.Append(BeginList());

 //result.Append(MakeItem("Temperature", MakeStatistics(data.Select(z => z.Temperature)).ToString()));

 //result.Append(MakeItem("Humidity", MakeStatistics(data.Select(z => z.Humidity)).ToString()));

 result.Append(EndList());

 return result.ToString();

 }

 \*/

 }

 public class Calcualtions

 {

 public static List<Measurement> Data;

 public static IEnumerable<double> GetTemperature()

 {

 return Data.Select(z => z.Temperature);

 }

 public static IEnumerable<double> GetHumidity()

 {

 return Data.Select(z => z.Humidity);

 }

 public static double GetAverageTemperature()

 {

 return Data.Select(z => z.Temperature).Average();

 }

 public static double GetAverageHumidity()

 {

 return Data.Select(z => z.Humidity).Average();

 }

 public static double GetTemperatureStandartDeviation()

 {

 return Math.Sqrt(Data.Select(z => Math.Pow(z.Temperature - GetAverageTemperature(), 2)).Sum() / (Data.Count - 1));

 }

 public static double GetHumidityStandartDeviation()

 {

 return Math.Sqrt(Data.Select(z => Math.Pow(z.Humidity - GetAverageHumidity(), 2)).Sum() / (Data.Count - 1));

 }

 private static double GetMedian(IEnumerable<double> source)

 {

 // Create a copy of the input, and sort the copy

 double[] temp = source.ToArray();

 Array.Sort(temp);

 int count = temp.Length;

 if (count == 0)

 {

 throw new InvalidOperationException("Empty collection");

 }

 else if (count % 2 == 0)

 {

 // count is even, average two middle elements

 double a = temp[count / 2 - 1];

 double b = temp[count / 2];

 return (a + b) / 2.0;

 }

 else

 {

 // count is odd, return the middle element

 return temp[count / 2];

 }

 }

 public static double GetHumidityMedian()

 {

 return GetMedian(Data.Select(z => z.Humidity));

 }

 public static double GetTemperatureMedian()

 {

 return GetMedian(Data.Select(z => z.Temperature));

 }

 }

 public class Printing{

 public static string PrintMeanAndStd(ReportMaker printer, List<Measurement> pData)

 {

 Calcualtions.Data = pData;

 double AvTmp = Calcualtions.GetAverageTemperature();

 double AvHum = Calcualtions.GetAverageHumidity();

 double DevTmp = Calcualtions.GetTemperatureStandartDeviation();

 double DevHum = Calcualtions.GetHumidityStandartDeviation();

 var result = new StringBuilder();

 result.Append(printer.MakeCaption("Mean and Std"));

 result.Append(printer.BeginList());

 result.Append(printer.MakeItem("Temperature",String.Format(CultureInfo.InvariantCulture, "{0}±{1}", AvTmp, DevTmp)));

 result.Append(printer.MakeItem("Humidity", String.Format(CultureInfo.InvariantCulture, "{0}±{1}", AvHum, DevHum)));

 result.Append(printer.EndList());

 return result.ToString();

 }

 public static string PrintMedian(ReportMaker printer, List<Measurement> pData)

 {

 Calcualtions.Data = pData;

 double MedTmp = Calcualtions.GetTemperatureMedian();

 double MedHum = Calcualtions.GetHumidityMedian();

 var result = new StringBuilder();

 result.Append(printer.MakeCaption("Median"));

 result.Append(printer.BeginList());

 result.Append(printer.MakeItem("Temperature", String.Format(CultureInfo.InvariantCulture,"{0}", MedTmp)));

 result.Append(printer.MakeItem("Humidity", String.Format(CultureInfo.InvariantCulture,"{0}", MedHum)));

 result.Append(printer.EndList());

 return result.ToString();

 }

 }

 public class HtmlReportMaker : ReportMaker

 {

 public override string Caption

 {

 get

 {

 return "Mean and Std";

 }

 }

 public override string MakeCaption(string caption)

 {

 return $"<h1>{caption}</h1>";

 }

 public override string BeginList()

 {

 return "<ul>";

 }

 public override string EndList()

 {

 return "</ul>";

 }

 public override string MakeItem(string valueType, string entry)

 {

 return $"<li><b>{valueType}</b>: {entry}";

 }

 }

 public class MarkdownReportMaker : ReportMaker

 {

 public override string Caption

 {

 get

 {

 return "Median";

 }

 }

 public override string BeginList()

 {

 return "";

 }

 public override string EndList()

 {

 return "";

 }

 public override string MakeCaption(string caption)

 {

 return $"## {caption}\n\n";

 }

 public override string MakeItem(string valueType, string entry)

 {

 return $" \* \*\*{valueType}\*\*: {entry}\n\n";

 }

 }

 public static class ReportMakerHelper

 {

 public static string MeanAndStdHtmlReport(IEnumerable<Measurement> data)

 {

 HtmlReportMaker hm = new HtmlReportMaker();

 return hm.MakeReport(Printing.PrintMeanAndStd, hm, data.ToList<Measurement>());

 }

 public static string MedianMarkdownReport(IEnumerable<Measurement> data)

 {

 MarkdownReportMaker mm = new MarkdownReportMaker();

 return mm.MakeReport(Printing.PrintMedian, mm, data.ToList<Measurement>());

 }

 public static string MeanAndStdMarkdownReport(IEnumerable<Measurement> measurements)

 {

 MarkdownReportMaker mm = new MarkdownReportMaker();

 return mm.MakeReport(Printing.PrintMeanAndStd, mm, measurements.ToList<Measurement>());

 }

 public static string MedianHtmlReport(IEnumerable<Measurement> measurements)

 {

 HtmlReportMaker hm = new HtmlReportMaker();

 return hm.MakeReport(Printing.PrintMedian, hm, measurements.ToList<Measurement>());

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Taymery__aac68121-b6a2-4f58-81ca-5d8dd934ab8c>

Давайте напишем удобную утилиту для внутриязыкового профайлинга. В реальности, конечно, так никто не делает, и используется внешний профайлер (например, dotTrace), но пусть.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Memory.Timers

{

 public class Pair : IComparable {

 public string Name;

 public double Value;

 public int NestingLevel;

 public Pair(string pName, double pValue, int pNestingLevel)

 {

 Name = pName;

 Value = pValue;

 NestingLevel = pNestingLevel;

 }

 public int CompareTo(object o)

 {

 Pair p = (Pair)o;

 return String.Compare(Name, p.Name);

 }

 }

 public static class ReportClass

 {

 public static List<Pair> Pairs = new List<Pair>();

 public static void Add(string pName, double pValue, int pNestingLevel)

 {

 Pairs.Add(new Pair(pName, pValue, pNestingLevel));

 }

 public static string GetSpaces(string p, int count)

 {

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

 {

 p += " ";

 }

 return p;

 }

 public static string GetSpacesBefore(string p, int count)

 {

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

 {

 p = " " + p;

 }

 return p;

 }

 public static double GetInnerTime(int outter)

 {

 double sum = 0;

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

 {

 if (Pairs[i].NestingLevel > outter)

 {

 sum += Pairs[i].Value;

 }

 }

 if (sum == 0)

 {

 sum = Pairs[outter].Value;

 }

 return sum;

 }

 public static string Get()

 {

 string s = "";

 Pairs.Sort();

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

 {

 int spaces = Pairs[i].NestingLevel \* 4;

 int spaces\_after = 20 - spaces - Pairs[i].Name.Length;

 if (Pairs[i].Name == " ")

 {

 spaces\_after -= 1;

 }

 string s\_spaces = GetSpacesBefore(Pairs[i].Name, spaces);

 string s\_spaces\_a = GetSpaces(s\_spaces, spaces\_after);

 //Console.WriteLine(Pairs[i].Value);

 //Console.WriteLine(GetInnerTime(Pairs[i].NestingLevel));

 string statement = String.Format("{0}: {1:0}\n", s\_spaces\_a, Pairs[i].Value);

 //string statement = String.Format("{0}: {1:0}\n", s\_spaces\_a, GetInnerTime(Pairs[i].NestingLevel));

 if (Pairs[i].Name == " " && Pairs.Count == 1)

 {

 statement = "\*" + statement;

 }

 //Console.WriteLine(statement);

 s += statement;

 }

 bool prev\_level\_diff = false;

 for (int i = Pairs.Count-1; i > 0; i--)

 {

 if (Pairs[i-1].NestingLevel <= Pairs[i].NestingLevel || (i == Pairs.Count - 1 && Pairs[i].NestingLevel != 0))

 {

 prev\_level\_diff = true;

 } else

 {

 prev\_level\_diff = false;

 }

 if (prev\_level\_diff == true)

 {

 int spaces = Pairs[i].NestingLevel \* 4;

 int spaces\_after = 20 - spaces - "Rest".Length;

 string s\_spaces = GetSpacesBefore("Rest", spaces);

 string s\_spaces\_a = GetSpaces(s\_spaces, spaces\_after);

 double v = Pairs[i].Value;

 if (Pairs[i - 1].NestingLevel <= Pairs[i].NestingLevel)

 {

 v += Pairs[i - 1].Value;

 }

 string statement = String.Format("{0}: {1:0}\n", s\_spaces\_a, v);

 //Console.WriteLine(statement);

 s += statement;

 }

 }

 Pairs.Clear();

 return s;

 }

 }

 public class Timer : IDisposable

 {

 public string Name { get; set; }

 public DateTime StartTime;

 public static int NestingLevel = 0;

 public static Timer Start(string pName = " ")

 {

 return new Timer(pName);

 }

 public Timer(string pName)

 {

 StartTime = DateTime.Now;

 Name = pName;

 NestingLevel += 1;

 }

 public static string Report

 {

 get

 {

 return ReportClass.Get();

 }

 }

 public void Dispose()

 {

 NestingLevel -= 1;

 ReportClass.Add(Name, (DateTime.Now - StartTime).Milliseconds, NestingLevel);

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_ValueType__a24f086b-c417-4f26-8833-c22897eff0c7>

Скачайте проект [Ddd.Taxi](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=a24f086b-c417-4f26-8833-c22897eff0c7).

Все Value-типы, согласно DDD, должны поддерживать семантику значений, то есть сравниваться по содержимому своих свойств. Каждый раз реализовывать Equals, GetHashCode и ToString соответствующим образом — довольно муторное занятие. Часто для этого создают базовый класс, наследование от которого реализует нужным образом все эти стандартные методы. Это вам и предстоит сделать!

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using Ddd.Taxi.Domain;

using System.Security.Cryptography;

using System.Text;

namespace Ddd.Infrastructure

{

 public class MatchingPair

 {

 public string PropertyName = System.String.Empty;

 public bool PropertyMatch = false;

 }

 /// <summary>

 /// Базовый класс для всех Value типов.

 /// </summary>

 public class ValueType<T>

 {

 public bool Equals(PersonName name)

 {

 return false;

 }

 private static int counter = 500;

 public override int GetHashCode()

 {

 string sSourceData;

 byte[] tmpSource;

 byte[] tmpHash;

 sSourceData = this.ToString();

 //Create a byte array from source data.

 tmpSource = ASCIIEncoding.ASCII.GetBytes(sSourceData);

 tmpHash = new MD5CryptoServiceProvider().ComputeHash(tmpSource);

 int a = BitConverter.ToInt32(tmpHash, 0);

 int b = BitConverter.ToInt32(tmpHash, 4);

 int c = BitConverter.ToInt32(tmpHash, 8);

 int d = BitConverter.ToInt32(tmpHash, 12);

 return a;

 /\*

 int result = 1;

 Type a = this.GetType();

 int Count = a.GetProperties().ToList<PropertyInfo>().Count;

 int index\_of\_property = 1;

 foreach (var property in a.GetProperties())

 {

 PropertyInfo a\_info = a.GetProperty(property.Name);

 var a\_value = a\_info.GetValue(this);

 index\_of\_property += 1;

 if (a\_value == null)

 {

 continue;

 }

 Type a\_type = a\_value.GetType();

 try

 {

 var inner\_code = (int)a\_type.GetMethod("GetHashCode").Invoke(a\_value, new object[] { });

 var mn = (inner\_code + index\_of\_property) \* 3 / 150;

 result += (inner\_code + Count \* index\_of\_property) + mn;

 }

 catch

 {

 }

 }

 return result;

 \*/

 }

 public override string ToString()

 {

 Type a = this.GetType();

 List<string> propertys\_names = new List<string>();

 string result = a.Name + "(";

 foreach (var property in a.GetProperties(BindingFlags.Public | BindingFlags.Instance))

 {

 propertys\_names.Add(property.Name);

 }

 propertys\_names.Sort();

 for (int i = 0; i < propertys\_names.Count; i++)

 {

 PropertyInfo a\_info = a.GetProperty(propertys\_names[i]);

 var a\_value = a\_info.GetValue(this);

 if (a\_value != null) {

 result += propertys\_names[i] + ": " + a\_value.ToString();

 } else

 {

 result += propertys\_names[i] + ": " ;

 }

 if (i != propertys\_names.Count - 1)

 {

 result += "; ";

 }

 }

 result += ")";

 return result;

 }

 public override bool Equals(object obj)

 {

 // this - объект, для которого вызывается метод сравнения

 // obj - объект, с которым сравниваем

 if (obj == null)

 {

 return false;

 }

 Type a = this.GetType();

 Type b = obj.GetType();

 if (a != b)

 {

 return false;

 }

 int CountA = a.GetProperties().ToList<PropertyInfo>().Count;

 int CountB = b.GetProperties().ToList<PropertyInfo>().Count;

 if (CountA != CountB)

 {

 return false;

 }

 List< MatchingPair > matching = new List<MatchingPair>();

 // Проверим свойства объектов

 foreach (var property\_a in a.GetProperties())

 {

 PropertyInfo a\_info = a.GetProperty(property\_a.Name);

 var a\_value = a\_info.GetValue(this);

 PropertyInfo b\_info = b.GetProperty(property\_a.Name);

 // Если во втором объекте не найдено свойство, которое есть в первом, то он нам не подходит

 if (b\_info == null)

 {

 return false;

 }

 var b\_value = b\_info.GetValue(obj);

 if (a\_value == null && b\_value != null)

 {

 return false;

 }

 if (a\_value != null && b\_value == null)

 {

 return false;

 }

 if (a\_value == null && b\_value == null)

 {

 continue;

 }

 try

 {

 // Если получится - приведем к IComparable

 IComparable a\_value\_c = (IComparable)a\_value;

 IComparable b\_value\_c = (IComparable)b\_value;

 // Если свойство есть у обоих объектов, но они не равны, то объекты не равны

 if (a\_value\_c.CompareTo(b\_value\_c) != 0)

 {

 return false;

 }

 }

 catch(Exception e)

 {

 // Если не удается привести к IComparable

 Type a\_value\_type = a\_value.GetType();

 Type b\_value\_type = b\_value.GetType();

 if (a\_value\_type != b\_value\_type)

 {

 return false;

 }

 // Если сравниваются строки

 if (a\_value\_type == typeof(string))

 {

 if (String.Compare((string)a\_value, (string)b\_value) != 0)

 {

 return false;

 }

 }

 // Если свойство имеет тип класса, который наследуется от этого

 if (a\_value\_type == typeof(PersonName))

 {

 if (a\_value.Equals(b\_value) == false)

 {

 return false;

 }

 }

 // Если свойство имеет тип DateTime

 if (a\_value\_type == typeof(DateTime))

 {

 if ((DateTime)a\_value != (DateTime)b\_value)

 {

 return false;

 }

 }

 }

 }

 return true;

 }

 }

}

<https://ulearn.me/Course/cs2/Praktika_TaxiOrder__4c09a41e-51aa-4f4d-b458-d3359ce7a622>

Продолжайте в том же проекте [Ddd.Taxi](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=4c09a41e-51aa-4f4d-b458-d3359ce7a622).

Изучите пару классов TaxiOrder и TaxiApi — это модель предметной области по заказу такси.

TaxiOrder — типичная анемичная модель. Вся логика, связанная с этим классом находится в TaxiApi.

Переработайте класс TaxiOrder согласно принципам DDD.

# Решение

using System;

using System.Globalization;

using System.Collections.Generic;

using System.Linq;

using Ddd.Infrastructure;

namespace Ddd.Taxi.Domain

{

 // In real aplication it whould be the place where database is used to find driver by its Id.

 // But in this exercise it is just a mock to simulate database

 public class DriversRepository

 {

 public void FillDriverToOrder(int driverId, TaxiOrder order)

 {

 if (driverId == 15)

 {

 order.SetData(new Driver(driverId, new PersonName("Drive", "Driverson")), "Lada sedan", "Baklazhan", "A123BT 66");

 }

 else

 throw new Exception("Unknown driver id " + driverId);

 }

 }

 public class TaxiApi : ITaxiApi<TaxiOrder>

 {

 private readonly DriversRepository driversRepo;

 private readonly Func<DateTime> currentTime;

 private int idCounter;

 public TaxiApi(DriversRepository driversRepo, Func<DateTime> currentTime)

 {

 this.driversRepo = driversRepo;

 this.currentTime = currentTime;

 }

 public TaxiOrder CreateOrderWithoutDestination(string firstName, string lastName, string street, string building)

 {

 return

 new TaxiOrder

 (

 idCounter++,

 new PersonName(firstName, lastName),

 new Address(street, building),

 currentTime

 );

 }

 public void UpdateDestination(TaxiOrder order, string street, string building)

 {

 order.UpdateDestination(new Address(street, building));

 }

 public void AssignDriver(TaxiOrder order, int driverId)

 {

 order.AssignDriver();

 driversRepo.FillDriverToOrder(driverId, order);

 }

 public void UnassignDriver(TaxiOrder order)

 {

 order.UnassignDriver();

 }

 public string GetDriverFullInfo(TaxiOrder order)

 {

 if (order.GetStatus == TaxiOrderStatus.WaitingForDriver) return null;

 return order.GetDriverFullInfo();

 }

 public string GetShortOrderInfo(TaxiOrder order)

 {

 return order.GetShortOrderInfo();

 }

 private DateTime GetLastProgressTime(TaxiOrder order)

 {

 return order.GetLastProgressTime();

 }

 private string FormatName(string firstName, string lastName)

 {

 return string.Join(" ", new[] { firstName, lastName }.Where(n => n != null));

 }

 private string FormatAddress(string street, string building)

 {

 return string.Join(" ", new[] { street, building }.Where(n => n != null));

 }

 public void Cancel(TaxiOrder order)

 {

 order.Cancel();

 }

 public void StartRide(TaxiOrder order)

 {

 order.StartRide();

 }

 public void FinishRide(TaxiOrder order)

 {

 order.FinishRide();

 }

 }

 public class Driver : Entity<int>

 {

 public int Id;

 public PersonName Name;

 private Car \_Car;

 public Car Car{

 get

 {

 return \_Car;

 }

 set

 {

 \_Car = value;

 }

 }

 public Driver(int pId, PersonName pName) : base(pId)

 {

 Id = pId;

 Name = pName;

 }

 }

 public class Car : ValueType<Car>

 {

 public string CarColor;

 public string CarModel;

 public string CarPlateNumber;

 public Car(string pCarColor, string pCarModel, string pCarPlateNumber)

 {

 CarColor = pCarColor;

 CarModel = pCarModel;

 CarPlateNumber = pCarPlateNumber;

 }

 }

 public class TaxiOrder : Entity<int>

 {

 private int Id;

 private PersonName \_ClientName;

 private Address \_Start;

 private Address \_Destination;

 private Driver \_Driver;

 private TaxiOrderStatus Status;

 private DateTime CreationTime;

 private DateTime DriverAssignmentTime;

 private DateTime CancelTime;

 private DateTime StartRideTime;

 private DateTime FinishRideTime;

 private Func<DateTime> currentTime;

 // Привет, глючный тест

 public PersonName ClientName { get; }

 public Address Start { get; }

 public Address Destination { get; }

 public Driver Driver { get; }

 public TaxiOrderStatus GetStatus

 {

 get

 {

 return Status;

 }

 }

 public void SetData(Driver pDriver, string pCarModel, string pCarColor, string pCarPlateNumber)

 {

 \_Driver = pDriver;

 \_Driver.Car = new Car(pCarColor, pCarModel, pCarPlateNumber);

 }

 public TaxiOrder(int pId, PersonName pClientName, Address pStart, Func<DateTime> pCurrentTime) : base(pId)

 {

 Id = pId;

 \_ClientName = pClientName;

 \_Start = pStart;

 CreationTime = pCurrentTime();

 currentTime = pCurrentTime;

 }

 public void UpdateDestination(Address p)

 {

 \_Destination = p;

 }

 public void AssignDriver()

 {

 if (\_Driver != null)

 {

 throw new System.InvalidOperationException("Blablabla");

 }

 DriverAssignmentTime = currentTime();

 Status = TaxiOrderStatus.WaitingCarArrival;

 }

 public void UnassignDriver()

 {

 if (Status == TaxiOrderStatus.InProgress || Status == TaxiOrderStatus.Finished)

 {

 throw new System.InvalidOperationException("Нельзя снять водителя с заказа после начала поездки");

 }

 if (\_Driver == null)

 {

 throw new System.InvalidOperationException("WaitingForDriver");

 }

 \_Driver = null;

 Status = TaxiOrderStatus.WaitingForDriver;

 }

 public string GetDriverFullInfo()

 {

 if (Status == TaxiOrderStatus.WaitingForDriver) return null;

 return string.Join(" ",

 "Id: " + \_Driver.Id,

 "DriverName: " + FormatName((\_Driver == null) ? null : \_Driver.Name),

 "Color: " + \_Driver.Car.CarColor,

 "CarModel: " + \_Driver.Car.CarModel,

 "PlateNumber: " + \_Driver.Car.CarPlateNumber);

 }

 public string GetShortOrderInfo()

 {

 return string.Join(" ",

 "OrderId: " + Id,

 "Status: " + Status,

 "Client: " + FormatName(\_ClientName),

 "Driver: " + FormatName((\_Driver == null) ? null : \_Driver.Name),

 "From: " + FormatAddress(\_Start),

 "To: " + FormatAddress(\_Destination),

 "LastProgressTime: " + GetLastProgressTime().ToString("yyyy-MM-dd HH:mm:ss", CultureInfo.InvariantCulture));

 }

 public DateTime GetLastProgressTime()

 {

 if (Status == TaxiOrderStatus.WaitingForDriver) return CreationTime;

 if (Status == TaxiOrderStatus.WaitingCarArrival) return DriverAssignmentTime;

 if (Status == TaxiOrderStatus.InProgress) return StartRideTime;

 if (Status == TaxiOrderStatus.Finished) return FinishRideTime;

 if (Status == TaxiOrderStatus.Canceled) return CancelTime;

 throw new NotSupportedException(Status.ToString());

 }

 private string FormatName(PersonName p)

 {

 return (p == null) ? "" : p.FirstName + " " + p.LastName;

 }

 private string FormatAddress(Address p)

 {

 return (p == null) ? "" : p.Street + " " + p.Building;

 }

 public void Cancel()

 {

 if (Status == TaxiOrderStatus.InProgress || Status == TaxiOrderStatus.Finished)

 {

 throw new System.InvalidOperationException("Нельзя отменить заказ после начала поездки");

 }

 Status = TaxiOrderStatus.Canceled;

 CancelTime = currentTime();

 }

 public void StartRide()

 {

 if (\_Driver == null)

 {

 throw new System.InvalidOperationException("Нельзя начать поездку, если у тебя нет водителя (мем бы сюда)");

 }

 Status = TaxiOrderStatus.InProgress;

 StartRideTime = currentTime();

 }

 public void FinishRide()

 {

 if (\_Driver == null)

 {

 throw new System.InvalidOperationException("Нельзя завершить поездку, если у тебя нет водителя (и сюда тоже)");

 }

 if (Status != TaxiOrderStatus.InProgress)

 {

 throw new System.InvalidOperationException("Нельзя завершить поездку до ее начала");

 }

 Status = TaxiOrderStatus.Finished;

 FinishRideTime = currentTime();

 }

 }

}

# <https://ulearn.me/Course/cs2/Sdelat_Fluent_1a6dfb60-84ee-4a76-b48a-c75956dfaa12>

Сделайте интерфейс работы с List чуть более Fluent. Создайте extension-метод AddItems, чтобы следующий код работал!

# Решение

 public static List<T> AddItems<T>(this List<T> items, params T[] p)

 {

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

 items.Add(p[i]);

 }

 return items;

 }

# <https://ulearn.me/Course/cs2/Praktika_Analiz_par__68cb2552-6413-456b-8269-2000d21ac9c9>

Делегаты позволяют нам в ряде случаев обходится вообще без создания классов. Скачайте проект [Delegates.PairsAnalysis](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=68cb2552-6413-456b-8269-2000d21ac9c9).

Метод Analysis.FindMaxPeriodIndex(params DateTime[] data) должен последовательно разбить список дат по парам и вернуть индекс пары с наибольшим периодом между датами.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace Delegates.PairsAnalysis

{

 public static class ListExtensions

 {

 public static IEnumerable<Tuple<T, T>> Pairs<T>(this IEnumerable<T> p)

 {

 List<Tuple<T, T>> result = new List<Tuple<T, T>>();

 IEnumerator<T> i = p.GetEnumerator();

 i.MoveNext();

 T Prev = i.Current;

 while (i.MoveNext())

 {

 //result.Add(new Tuple<T, T>(Prev, i.Current));

 yield return new Tuple<T, T>(Prev, i.Current);

 Prev = i.Current;

 }

 yield break;

 //return result.ToArray();

 }

 public static int MaxIndex<T>(this IEnumerable<Tuple<T, T>> p) where T : IComparable

 {

 int ind = -1;

 int int\_diff;

 int int\_max\_diff = int.MinValue;

 TimeSpan dt\_diff;

 double dt\_max\_diff = double.MinValue;

 Type type = typeof(T);

 IEnumerator<Tuple<T, T>> i = p.GetEnumerator();

 int counter = 0;

 while (i.MoveNext())

 {

 if (type == typeof(int))

 {

 int a = Convert.ToInt32(i.Current.Item1);

 int b = Convert.ToInt32(i.Current.Item2);

 int\_diff = (a > b) ? a - b : b - a;

 if (int\_diff > int\_max\_diff)

 {

 int\_max\_diff = int\_diff;

 ind = counter;

 }

 } else

 {

 if (type == typeof(DateTime))

 {

 DateTime a = Convert.ToDateTime(i.Current.Item1);

 DateTime b = Convert.ToDateTime(i.Current.Item2);

 dt\_diff = (a > b) ? a - b : b - a;

 if (dt\_diff.TotalDays > dt\_max\_diff)

 {

 dt\_max\_diff = dt\_diff.TotalDays;

 ind = counter;

 }

 } else

 {

 throw new NotImplementedException();

 }

 }

 counter += 1;

 }

 return ind;

 }

 public static int MaxIndex<T>(this IEnumerable<T> p) where T : IComparable

 {

 int ind = -1;

 int int\_diff;

 int int\_max\_diff = int.MinValue;

 Type type = typeof(T);

 IEnumerator<T> i = p.GetEnumerator();

 int counter = 0;

 int max = int.MinValue;

 while (i.MoveNext())

 {

 if (type == typeof(int))

 {

 int a = Convert.ToInt32(i.Current);

 if (a > max)

 {

 max = a;

 ind = counter;

 }

 } else

 {

 throw new NotImplementedException();

 }

 counter += 1;

 }

 if (ind == -1)

 {

 throw new System.ArgumentException("Передана пустая последовательность");

 }

 return ind;

 }

}

 public static class Analysis

 {

 public static int FindMaxPeriodIndex(params DateTime[] data)

 {

 if (data.Length == 0)

 {

 throw new ArgumentException("Не переданы данные для обработки");

 }

 if (data.Length == 1)

 {

 throw new ArgumentException("Передана одна дата. Невозможно образовать пару");

 }

 List<DateTime> dates = data.ToList<DateTime>();

 IEnumerable<Tuple<DateTime, DateTime>> pairs = dates.Pairs();

 return pairs.MaxIndex();

 }

 public static double FindAverageRelativeDifference(params double[] data)

 {

 return new AverageDifferenceFinder().Analyze(data);

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_API__4cd8c33d-f190-4889-a84b-d0d3abb7bea7>

Используем паттерн Disposable в ситуации, схожей с работой со внешним API. В [проекте Memory.API](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=4cd8c33d-f190-4889-a84b-d0d3abb7bea7) в роли внешнего API выступает класс MagicAPI, методы которого позволяют выделить ресурс, освободить его, и проверить, какие ресурсы выделены в настоящий момент.

Реализуйте класс APIObject, который будет оберткой над API. Выделение ресурса через внешнее API должна происходить в конструкторе, а освобождение - в соответствие с паттерном Disposable.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Memory.API

{

 public class APIObject : IDisposable

 {

 private bool isDisposed = false;

 public int Id;

 public APIObject(int pId)

 {

 Id = pId;

 MagicAPI.Allocate(pId);

 }

 ~APIObject()

 {

 Dispose(false);

 }

 public void Dispose()

 {

 Dispose(true);

 GC.SuppressFinalize(this);

 }

 public void Dispose(bool CalledFromDisposeMethod)

 {

 if (!isDisposed)

 {

 MagicAPI.Free(Id);

 isDisposed = true;

 }

 }

 }

}

<https://ulearn.me/Course/cs2/Praktika_Resursy__d0d5a681-60f1-45b4-a066-17842e13d753>

Необходимость писать собственные стримы бывает не так уж и часто. Однако, такие ситуации бывают.

Например, допустим, что вы разрабатываете компьютерную игру с множеством мелких файлов. Очевидно, что хотелось бы эти файлы убрать в один. Допустим, что вы по какой-то причине не хотите использовать zip-сжатие (что было бы самым адекватным подходом к этой ситуации), и вместо этого хотите изобрести свой формат.

Ваша задача — по известному формату написать стрим, который читает секцию файла. Ваш стрим будет получать другой, базовый стрим, который содержит данные, и ваша задача — найти нужную секцию и прочитать. Эта задача осмысленна, поскольку, например, Bitmap.FromStream принимает именно Stream, и вы можете подставить туда ваш стрим для того, чтобы все работало.

# Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using System.Text;

using System.Threading.Tasks;

namespace Streams.Resources

{

 public class ResourceReaderStream : Stream

 {

 private TestStream UnderlayingStream;

 private string Key;

 private byte[] KeyByte;

 private static bool BufferStealed = false;

 private List<byte> Data = new List<byte>();

 private static string[] pairs = new string[0];

 private string Information;

 private bool InfoFound = false;

 private int InfoOffset = 0;

 public ResourceReaderStream(Stream pStream, string pKey)

 {

 UnderlayingStream = (TestStream)pStream;

 Key = pKey;

 KeyByte = Encoding.Default.GetBytes(Key);

 }

 public override int Read(byte[] buffer, int offset, int count)

 {

 if (BufferStealed == false)

 {

 BufferStealed = true;

 byte[] data\_tmp = new byte[Constants.BufferSize];

 while (true)

 {

 int check = UnderlayingStream.Read(data\_tmp, 0, Constants.BufferSize);

 if (check == 0)

 {

 break;

 }

 Data.AddRange(data\_tmp);

 }

 string data = Encoding.ASCII.GetString(Data.ToArray());

 string divider = Encoding.ASCII.GetString(new byte[] { 0, 1 });

 pairs = data.Split(new string[] { divider }, StringSplitOptions.RemoveEmptyEntries);

 }

 if (InfoFound == false)

 {

 for (int i = 0; i < pairs.Length - 2; i += 2)

 {

 string k = pairs[i];

 string v = pairs[i + 1];

 if (k == Key)

 {

 Information = v;

 InfoFound = true;

 break;

 }

 }

 if (InfoFound == false)

 {

 return 0;

 } else

 {

 // Начинаем писать

 int start = InfoOffset;

 int finish = (Information.Length > count) ? count : Information.Length;

 if (start + finish > Information.Length)

 {

 return 0;

 }

 int plus\_start = start;

 int plus = plus\_start;

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

 {

 try

 {

 if ((byte)Information[i + plus] == 0)

 {

 plus += 1;

 }

 else

 {

 buffer[i] = (byte)Information[i + plus];

 }

 }

 catch

 {

 }

 }

 int diff = plus\_start - plus;

 InfoOffset += finish;

 return finish + diff;

 }

 } else

 {

 // Дописываем

 int start = InfoOffset;

 int finish = (Information.Length > count) ? count : Information.Length;

 if (start >= Information.Length - 1)

 {

 return 0;

 }

 int plus\_start = start;

 int plus = plus\_start;

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

 {

 if ((byte)Information[i + plus] == 0)

 {

 plus += 1;

 }

 else

 {

 buffer[i] = (byte)Information[i + plus];

 }

 }

 int diff = plus\_start - plus;

 InfoOffset += finish;

 return finish + diff;

 }

 }

 public static int SeekValue(byte[] array, byte[] subArray)

 {

 int index = -1;

 for (int i = 0; i < array.Length - subArray.Length + 1; i++)

 {

 index = i;

 for (int j = 0; j < subArray.Length; j++)

 {

 if (array[i + j] != subArray[j])

 {

 index = -1;

 break;

 }

 }

 if (index >= 0)

 return index;

 }

 return -1;

 }

 public override bool CanRead

 {

 get

 {

 return true;

 }

 }

 public override bool CanSeek

 {

 get

 {

 throw new NotImplementedException();

 }

 }

 public override bool CanWrite

 {

 get

 {

 throw new NotImplementedException();

 }

 }

 public override long Length

 {

 get

 {

 throw new NotImplementedException();

 }

 }

 public override long Position

 {

 get

 {

 throw new NotImplementedException();

 }

 set

 {

 throw new NotImplementedException();

 }

 }

 public override void Flush()

 {

 throw new NotImplementedException();

 }

 public override long Seek(long offset, SeekOrigin origin)

 {

 throw new NotImplementedException();

 }

 public override void SetLength(long value)

 {

 throw new NotImplementedException();

 }

 public override void Write(byte[] buffer, int offset, int count)

 {

 throw new NotImplementedException();

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Kontrol_nyy_razryad__b3e33f3d-a2ad-45b9-929a-1e6ecb18006f>

Скачайте проект [SRP.ControlDigit](https://ulearn.me/Exercise/StudentZip?courseId=cs2&slideId=b3e33f3d-a2ad-45b9-929a-1e6ecb18006f).

В серийные номера, номера счетов и коды продуктов обычно включают так называемый контрольный разряд, значение которого вычисляется по остальным цифрам номера. Он нужен, чтобы подтверждать отсутствие ошибок при вводе этих номеров вручную или при считывании их с помощью сканеров.

Есть несколько стандартизированных алгоритмов вычисления контрольного разряда. Прочитать их краткое описание с примерами можно в соответствующей [статье википедии](https://en.wikipedia.org/wiki/Check_digit).

В проекте уже реализован один метод UPC. Причем реализована без какой-либо декомпозиции. Вам нужно реализовать две других функции, но это не главное. Главное, провести декомпозицию имеющегося кода, разбив его на небольшие повторно используемые функции.

В вашем решении, в классе ControlDigitAlgo должны остаться всего три коротких метода — по одному на каждый алгоритм рассчёта. Все остальные функции должны быть полезны, понятны и самодостаточны вне контекста этой задачи и находится в виде методов расширения в классе Extensions.

В этой задаче действует ограничение на длину методов — не более 10 строк. Но не надо пытаться записывать сразу много операторов в одной строке — это дурной тон в программировании.

После окончания ещё раз посмотрите на сигнатуры всех вспомогательных методов. Точно ли они будут понятны другому программисту без подглядывания в их код?

Считайте, что эти методы не будут вызываться слишком часто, поэтому не нужно пытаться их оптимизировать, вместо этого сосредоточьтесь на простоте, понятности и возможности повторного использования.

# Решение

using System;

namespace SRP.ControlDigit

{

 public static class Extensions

 {

 public static string Append(this string s, int count)

 {

 // Дополнить строку нулями слева до определенной длины

 int p = count - s.Length;

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

 {

 s = "0" + s;

 }

 return s;

 }

 public static string AppendIfNeed(this string s, int count)

 {

 if (s.Length < count)

 {

 return s.Append(count);

 } else

 {

 return s;

 }

 }

 }

 public static class ControlDigitAlgo

 {

 public static int Upc(long number)

 {

 int sum = 0;

 int factor = 3;

 do

 {

 int digit = (int)(number % 10);

 sum += factor \* digit;

 factor = 4 - factor;

 number /= 10;

 }

 while (number > 0);

 int result = sum % 10;

 if (result != 0)

 result = 10 - result;

 return result;

 }

 public static char Isbn10(long number)

 {

 // Дополнение до 9 цифр

 string numbs = number.ToString().AppendIfNeed(9);

 int count = numbs.Length;

 double sum = 0;

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

 {

 int numb = Convert.ToInt32(numbs[i].ToString());

 //Console.WriteLine("[[{0}] \* [{1}]]",numb, count - i + 1);

 sum += numb \* (count - i + 1);

 }

 double div = sum / 11;

 double rem = sum % 11;

 if (rem == 0)

 return '0';

 else if (rem == 1)

 return 'X';

 else

 return (11 - rem).ToString().ToCharArray()[0];

 }

 public static int Isbn13(long number)

 {

 // Дополнение до 12 цифр

 string numbs = number.ToString().AppendIfNeed(12);

 int count\_chet = 0;

 int count\_nechet = 0;

 for (int i = 2; i < numbs.Length;i++)

 {

 if (i % 2 == 0)

 {

 //Сумма по чётным позициям.

 count\_chet += int.Parse(numbs[i].ToString());

 }

 else

 {

 //Сумма по нечётным позициям.

 count\_nechet += int.Parse(numbs[i].ToString());

 }

 }

 //Контрольная сумма и контрольный разряд.

 int control\_summ = count\_chet \* 3 + count\_nechet;

 int ostatok = control\_summ % 10;

 if (!(ostatok == 0))

 {

 ostatok = 10 - ostatok;

 }

 return Convert.ToInt32(ostatok);

 }

 }

}

# <https://ulearn.me/Course/cs2/Praktika_Szhatie__f80e5b7f-3192-4f58-827f-292e3d8c1f93>

В этом задании вам нужно написать стрим, осуществляющий сжатие и восстановление информации.

Бороться за высокое качество сжатия не обязательно: стрим будет тестироваться на байтовых последовательностях, состоящих из 2-4-кратных повторений одного и того же байта.

Как и в предыдущем задании, при распаковке не следует читать сразу весь исходный поток. Однако, ограничений на размеры чтения нет.

using System;

using System.IO;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO.Compression;

namespace Streams.Compression

{

 public class CustomCompressionStream : Stream

 {

 MemoryStream UnderStream;

 static List<byte> Data = new List<byte>();

 static List<byte> Res = new List<byte>();

 bool Done = false;

 // True - распаковка, False - запаковка

 bool Mode;

 public CustomCompressionStream(MemoryStream s, bool p)

 {

 UnderStream = s;

 Mode = p;

 }

 public override int Read(byte[] buffer, int offset, int count)

 {

 byte[] res;

 if (Done == false) {

 GZipStream bigStream = new GZipStream(new MemoryStream(Data.ToArray()), CompressionMode.Decompress);

 MemoryStream bigStreamOut = new MemoryStream();

 bigStream.CopyTo(bigStreamOut);

 res = bigStreamOut.ToArray();

 Res = res.ToList();

 bigStream.Close();

 bigStreamOut.Close();

 Done = true;

 }

 res = Res.ToArray();

 int finish = (res.Length < count) ? res.Length : count;

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

 {

 buffer[i] = res[i];

 }

 Res = Res.Skip(finish).ToList<byte>();

 return finish;

 }

 public override void Write(byte[] buffer, int offset, int count)

 {

 MemoryStream inStream = new MemoryStream(buffer);

 MemoryStream outStream = new MemoryStream();

 using (GZipStream tinyStream = new GZipStream(outStream, CompressionMode.Compress))

 {

 inStream.CopyTo(tinyStream);

 }

 Data = outStream.ToArray().ToList<byte>();

 inStream.Close();

 outStream.Close();

 }

 public override bool CanRead

 {

 get

 {

 return true;

 }

 }

 public override bool CanSeek

 {

 get

 {

 throw new NotImplementedException();

 }

 }

 public override bool CanWrite

 {

 get

 {

 throw new NotImplementedException();

 }

 }

 public override long Length

 {

 get

 {

 throw new NotImplementedException();

 }

 }

 public override long Position

 {

 get

 {

 throw new NotImplementedException();

 }

 set

 {

 throw new NotImplementedException();

 }

 }

 public override void Flush()

 {

 throw new NotImplementedException();

 }

 public override long Seek(long offset, SeekOrigin origin)

 {

 throw new NotImplementedException();

 }

 public override void SetLength(long value)

 {

 throw new NotImplementedException();

 }

 }

}