



Модул 5. Обектно-ориентирано програмиране

Тема 1. Компонентно тестване

Задача 1.1. Скелет

Създайте нов конзолен проект наименуван **Skeleton** и съдържащ следните три класа:

Axe.cs

```
public class Axe
{
    private int attackPoints;
    private int durabilityPoints;

    public Axe(int attack, int durability)
    {
        this.attackPoints = attack;
        this.durabilityPoints = durability;
    }

    public int AttackPoints
    {
        get { return this.attackPoints; }
    }

    public int DurabilityPoints
    {
        get { return this.durabilityPoints; }
    }

    public void Attack(Dummy target)
    {
        if (this.durabilityPoints <= 0)
        {
            throw new InvalidOperationException("Axe is broken.");
        }

        target.TakeAttack(this.attackPoints);
        this.durabilityPoints -= 1;
    }
}
```

Dummy.cs

```
public class Dummy
{
    private int health;
    private int experience;

    public Dummy(int health, int experience)
    {
        this.health = health;
        this.experience = experience;
    }
}
```



```
public int Health
{
    get { return this.health; }
}

public void TakeAttack(int attackPoints)
{
    if (this.IsDead())
    {
        throw new InvalidOperationException("Dummy is dead.");
    }

    this.health -= attackPoints;
}

public int GiveExperience()
{
    if (!this.IsDead())
    {
        throw new InvalidOperationException("Target is not dead.");
    }

    return this.experience;
}

public bool IsDead()
{
    return this.health <= 0;
}
}

Hero.cs
public class Hero
{
    private string name;
    private int experience;
    private Axe weapon;

    public Hero(string name)
    {
        this.name = name;
        this.experience = 0;
        this.weapon = new Axe(10, 10);
    }

    public string Name
    {
        get { return this.name; }
    }

    public int Experience
    {
        get { return this.experience; }
    }
}
```



```
public Axe Weapon
{
    get { return this.weapon; }
}

public void Attack(Dummy target)
{
    this.weapon.Attack(target);

    if (target.IsDead())
    {
        this.experience += target.GiveExperience();
    }
}
```

Задача 1.2. NUnit местове на Axe

Създайте нов проект наменен **NUnitTests**, като използвате шаблона **NUnit Test Project**. Уверете се, че имате включен **Reference** към **Skeleton** проекта. Ако нямате добавете такъв чрез десен бутон върху проекта **Tests > [Add] > [Reference]** и от там изберете **Solution > Skeleton**. Напишете метод наменен **AxeLosesDurablyAfterAttack**, който тества дали оръжието губи здравина след всяка атака. Напишете метод наменен **BrokenAxeCantAttack**, който тества атака със счупено оръжие.

Решение

```
namespace NUnitTests
{
    public class AxeTests
    {
        private const int AxeAttack = 2;
        private const int AxeDurability = 2;
        private const int DummyHealth = 20;
        private const int DummyXP = 20;

        private Axe axe;
        private Dummy dummy;

        [SetUp]
        public void TestInit()
        {
            this.axe = new Axe(AxeAttack, AxeDurability);
            this.dummy = new Dummy(DummyHealth, DummyXP);
        }

        [Test]
        public void AxeLosesDurablyAfterAttack()
        {
            axe.Attack(dummy);
            Assert.AreEqual(1, axe.DurabilityPoints, "Axe Durability doesn't change after attack");
        }
    }
}
```



```
    }

    [Test]
    public void BrokenAxeCantAttack()
    {
        axe.Attack(dummy);
        axe.Attack(dummy);
        var ex = Assert.Throws<InvalidOperationException>(() =>
axe.Attack(dummy));
        Assert.That(ex.Message, Is.EqualTo("Axe is broken.));
    }
}
```

Задача 1.3. MSTest местове на Axe

Създайте нов проект наименован **MicrosoftTests**, като използвате шаблона **MSTest Test Project**. Уверете се, че имате включен **Reference** към **Skeleton** проекта. Ако нямате добавете такъв чрез десен бутон върху проекта **Tests** > **[Add]** > **[Reference]** и от там изберете **Solution** > **Skeleton**. Напишете метод наименован **AxeLosesDurabilityAfterAttack**, който тества дали оръжието губи здравина след всяка атака. Напишете метод наименован **BrokenAxeCantAttack**, който тества атака със счупено оръжие.

Решение

```
namespace MicrosoftTests
{
    [TestClass]
    public class AxeTests
    {
        private const int AxeAttack = 2;
        private const int AxeDurability = 2;
        private const int DummyHealth = 20;
        private const int DummyXP = 20;

        private Axe axe = new Axe(AxeAttack, AxeDurability);
        private Dummy dummy = new Dummy(DummyHealth, DummyXP);

        [TestMethod]
        public void AxeLosesDurabilityAfterAttack()
        {
            axe.Attack(dummy);
            Assert.AreEqual(1, axe.DurabilityPoints, "Axe Durability doesn't change
after attack");
        }

        [TestMethod]
        public void BrokenAxeCantAttack()
        {
            axe.Attack(dummy);
            axe.Attack(dummy);
        }
    }
}
```



```
        var ex = Assert.ThrowsException<InvalidOperationException>(() =>
axe.Attack(dummy));
        Assert.AreEqual(ex.Message, "Axe is broken.");
    }
}
```

Задача 1.4. NUnit тестове на Dummy

Към проекта **NUnitTests**, добавете клас **DummyTests** и 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

Решение

```
namespace NUnitTests
{
    public class DummyTests
    {
        [Test]
        public void DummyLosesHealthWhenIsAttacked()
        {
            // Arrange
            Axe axe = new Axe(1, 10);
            Dummy dum = new Dummy(10, 10);

            // Act
            var oldHealth = dum.Health;
            axe.Attack(dum);

            // Assert
            Assert.AreNotSame(oldHealth, dum.Health);
        }

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

            // Act
            axe.Attack(dummy);

            // Assert
            Assert.Catch<InvalidOperationException>(delegate
            {
                axe.Attack(dummy); // Expected Exception here!
            });
        }

        [Test]
```



```
public void IsDeadDummyGiveXP()
{
    // Arrange
    Dummy dummy = new Dummy(1, 1);
    Hero hero = new Hero("Коняря");

    // Act
    var xp = hero.Experience;
    hero.Attack(dummy);
    xp = hero.Experience - xp;

    // Assert
    Assert.IsTrue(xp > 0);
}

[Test]
public void IsAliveDummyGiveXP()
{
    // Arrange
    Dummy dummy = new Dummy(100, 100);
    Hero hero = new Hero("Коняря");

    // Act
    var xp = hero.Experience;
    hero.Attack(dummy);
    xp = hero.Experience - xp;

    // Assert
    Assert.IsTrue(xp == 0);
}
}
```

Задача 1.5. MSTest местове на Dummy

Към проекта **MicrosoftTests** и добавете клас **DummyTests** и създайте следните тестове:

- Чуелото 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

Решение

```
namespace MicrosoftTests
{
    [TestClass]
    public class DummyTests
    {
        [TestMethod]
        public void DummyLosesHealthWhenIsAttacked()
        {
            // Arrange
            Axe axe = new Axe(1, 10);
            Dummy dum = new Dummy(10, 10);
```



```
// Act
var oldHealth = dum.Health;
axe.Attack(dum);

// Assert
Assert.AreNotSame(oldHealth, dum.Health);
}

[TestMethod]
[ExpectedException(typeof(InvalidOperationException), "Dummy is dead.")]

public void IsDeadDummyThrowsExceptionUnderAttack()
{
    // Arrange
    Axe axe = new Axe(10, 10);
    Dummy dummy = new Dummy(1, 1);

    // Act
    axe.Attack(dummy);
    axe.Attack(dummy); // Expected Exception here!

    // Assert
    Assert.IsTrue(dummy.Health > 0);
}

[TestMethod]
public void IsDeadDummyGiveXP()
{
    // Arrange
    Dummy dummy = new Dummy(1, 1);
    Hero hero = new Hero("Коняря");

    // Act
    var xp = hero.Experience;
    hero.Attack(dummy);
    xp = hero.Experience - xp;

    // Assert
    Assert.IsTrue(xp > 0);
}

[TestMethod]
public void IsAliveDummyGiveXP()
{
    // Arrange
    Dummy dummy = new Dummy(100, 100);
    Hero hero = new Hero("Коняря");

    // Act
    var xp = hero.Experience;
    hero.Attack(dummy);
    xp = hero.Experience - xp;

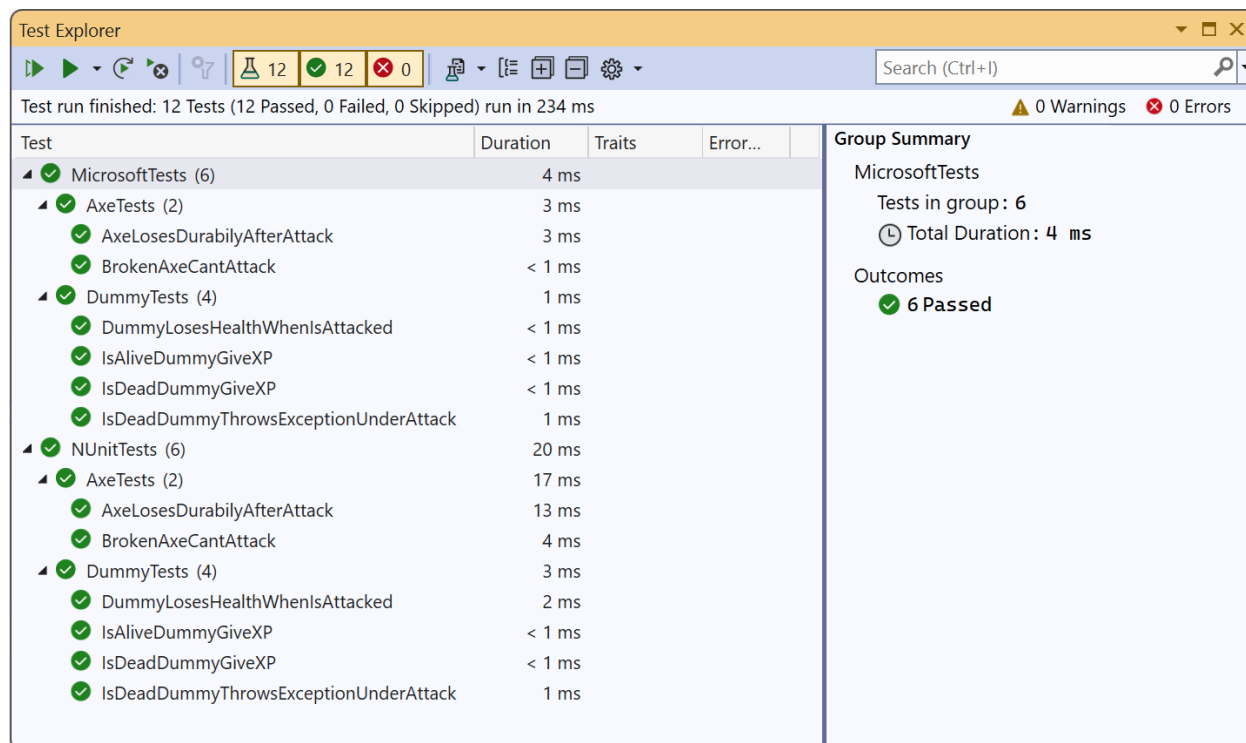
    // Assert
```



```
        Assert.IsTrue(xp == 0);  
    }  
}  
}
```

Бележка

Изпълнете тестовете от прозорчето **Test Explorer**. Резултатите трябва да изглеждат така:



Задача 1.6. Фалшиви Axe и Dummy

Тествайте дали героят получава XP, когато мишената умре. За целта трябва да:

- Направите класа Hero тестваем (използвайте Dependency Injection)
- Създайте Интерфейси за Axe и Dummy
- Интерфейс Weapon
- Интерфейс Target

Създайте фалшиви Weapon и Dummy за теста.

Подсказки

- Създайте IWeapon и ITarget интерфейси. Модифицирайте имплементационните методи, така че да ползват интерфейсите. Модифицирайте Axe и Dummy класовете.
- Използвайте Dependency Injection за Hero класа
- Създайте HeroTests клас и тествайте получаването на XP



Решение

Hero.cs

```
public class Hero
{
    private string name;
    private int experience;
    private IWeapon weapon;

    public Hero(string name, IWeapon weapon)
    {
        this.name = name;
        this.experience = 0;
        this.weapon = weapon;
    }

    public string Name
    {
        get { return this.name; }
    }

    public int Experience
    {
        get { return this.experience; }
    }

    public IWeapon Weapon
    {
        get { return this.weapon; }
    }

    public void Attack(ITarget target)
    {
        this.weapon.Attack(target);

        if (target.IsDead())
        {
            this.experience += target.GiveExperience();
        }
    }
}
```

ITarget.cs

```
public interface ITarget
{
    void TakeAttack(int attackPoints);
    int Health { get; }
    int GiveExperience();
    bool IsDead();
}
```

IWeapon.cs

```
public interface IWeapon
{
}
```



```
void Attack(ITarget target);  
int AttackPoints { get; }  
int DurabilityPoints { get; }  
}
```

FakeTarget.cs

```
public class FakeTarget : ITarget  
{  
    public int Health => 0;  
  
    public int GiveExperience() => 20;  
  
    public bool IsDead() => true;  
  
    public void TakeAttack(int attackPoints)  
    {  
        // Empty  
    }  
}
```

FakeWeapon.cs

```
public class FakeWeapon : IWeapon  
{  
    public int AttackPoints => 0;  
  
    public int DurabilityPoints => 0;  
  
    public void Attack(ITarget target)  
    {  
        // Empty  
    }  
}
```

HeroTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;  
  
namespace HeroTests  
{  
    [TestClass]  
    public class HeroTests  
    {  
        [TestMethod]  
        public void HeroGainsExperienceAfterAttackIfTargetIsDead()  
        {  
            // Arrange  
            ITarget fakeTarget = new FakeTarget();  
            IWeapon fakeWeapon = new FakeWeapon();  
            Hero hero = new Hero("Pesho", fakeWeapon);  
  
            // Act  
            hero.Attack(fakeTarget);  
  
            // Assert  
            Assert.IsTrue(hero.Experience > 0);  
        }  
    }  
}
```



```
}  
}  
}
```

Задача 1.7. Имитиране

Вкарайте Moq в проекта си и имитирайте фалшификатите от предишната задача. Използвайте **HeroTests** и рефакторирайте кода, така щото да ползвате Moq.

Решение

```
using Moq;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
  
namespace MockingTests  
{  
    [TestClass]  
    public class MockingTests  
    {  
        [TestMethod]  
        public void HeroGainsExperienceAfterAttackIfTargetIsDead()  
        {  
            // Arrange  
  
            // 1. Fake Target  
            Mock<ITarget> fakeTarget = new Mock<ITarget>();  
            fakeTarget.Setup(p => p.Health).Returns(0);  
            fakeTarget.Setup(p => p.GiveExperience()).Returns(20);  
            fakeTarget.Setup(p => p.IsDead()).Returns(true);  
  
            // 2. Fake Weapon  
            Mock<IWeapon> fakeWeapon = new Mock<IWeapon>();  
  
            // 3. Hero  
            Hero hero = new Hero("Pesho", fakeWeapon.Object);  
  
            // Act  
            hero.Attack(fakeTarget.Object);  
  
            // Assert  
            Assert.AreEqual(20, hero.Experience);  
        }  
    }  
}
```

Тема 2. Дефиниране на класове

Задача 2.1. Дефиниране на клас човек

Дефинирайте клас **Person** с public полета name (име) и age (възраст).

Забележка

Добавете следния код във вашият Main метод.

```
static void Main(string[] args)
```



```
{
    Type personType = typeof (Person);
    FieldInfo[] fields = personType.GetFields(BindingFlags.Public |
BindingFlags.Instance);
    Console.WriteLine(fields.Length);
}
```

Ако сте дефинирали класа правилно, тестът трябва да мине.

Бонус*

Опитайте се да създадете няколко обекта от Person:

Име	Възраст
Pesho	20
Gosho	18
Stamat	43

Решение

namespace Solution

```
{
    public class Person
    {
        private string name;

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

        private int age;

        public int Age
        {
            get { return age; }
            set { age = value; }
        }

        public void IntroduceYourself()
        {
            Console.WriteLine("Hello! My name is {0} and I am {1} years
old.", name, age);
        }
    }
}
```

Задача 2.2. Семейство

Създайте клас Person с полета name (име) и age (възраст). Създайте клас Family. В този клас трябва да има списък от хора. Напишете програма, която въвежда информация за N човека от едно семейство, след което изведете семейството по азбучен ред.



Примери

Вход	Изход
3	Pesho 3
Pesho 3	Gosho 4
Gosho 4	Annie 5
Annie 5	

Вход	Изход
5	Steve 10
Steve 10	Christopher 15
Christopher 15	Annie 4
Annie 4	Ivan 35
Ivan 35	Maria 34
Maria 34	

Бонус*

Опитайте се да създадете метод Print за класа Family

Решение

```
namespace Family
{
    public class Person
    {
        private string name;

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

        private int age;

        public int Age
        {
            get { return age; }
            set { age = value; }
        }

        public void IntroduceYourself()
        {
            Console.WriteLine("Hello! My name is {0} and I am {1} years old.", name,
age);
        }
    }

    public class Family
    {
        private List<Person> family = new List<Person>();

        public void Add(string name, int age)
        {
            family.Add(new Person { Name = name, Age = age });
        }

        public void Print()
        {
            family.ForEach(person => Console.WriteLine($"{person.Name}
{person.Age}"));
        }
    }
}
```



```
    }  
}  
  
public class Program  
{  
    static void Main(string[] args)  
    {  
        Family family = new Family();  
  
        int n = int.Parse(Console.ReadLine());  
        for (int i = 0; i < n; i++)  
        {  
            var line = Console.ReadLine().Split().ToArray();  
            family.Add(line[0], int.Parse(line[1]));  
        }  
  
        family.Print();  
    }  
}
```

Задача 2.3. Статистика

Използвайки класът Person, напишете програма, която въвежда от конзолата N реда информация за хора и отпечатва хората на възраст по-голяма от 30 години, сортирани по азбучен ред.

Примери

Вход	Изход
3 Pesho 12 Stamat 31 Ivan 48	Ivan - 48 Stamat - 31
5 Nikolai 33 Yordan 88 Tosho 22 Lyubo 44 Stanislav 11	Lyubo - 44 Nikolai - 33 Yordan - 88

Решение

```
namespace Statistics  
{  
    public class Person  
    {  
        private string name;  
  
        public string Name  
        {  
            get { return name; }  
            set { name = value; }  
        }  
    }  
}
```



```
        private int age;

        public int Age
        {
            get { return age; }
            set { age = value; }
        }

        public void IntroduceYourself()
        {
            Console.WriteLine("Hello! My name is {0} and I am {1} years old.", name,
age);
        }
    }

    public class Family
    {
        private List<Person> family = new List<Person>();

        public void Add(string name, int age)
        {
            family.Add(new Person { Name = name, Age = age });
        }

        public void Print()
        {
            var filer = family.Where(x => x.Age > 30).OrderBy(x => x.Name).ToList();
            filer.ForEach(person => Console.WriteLine($"{person.Name} -
{person.Age}"));
        }
    }

    public class Program
    {
        static void Main(string[] args)
        {
            Family family = new Family();

            int n = int.Parse(Console.ReadLine());
            for (int i = 0; i < n; i++)
            {
                var line = Console.ReadLine().Split().ToArray();
                family.Add(line[0], int.Parse(line[1]));
            }

            family.Print();
        }
    }
}
```



Задача 2.4. Дефиниране на клас Рационално число

Дефинирайте клас **RationalNumber** с `private` полета `numerator` (числител) и `denominator` (знаменател). Създайте 3 обекта и ги изведете във формат "Numerator/denominator {numerator}/{denominator}":

Примери

Вход	Изход
3 4 5 6 7 8	3/4 5/6 7/8

Решение

```
namespace RationalNumber
{
    public class RationalNumber
    {
        private int number;

        public int Number
        {
            get { return number; }
            set { number = value; }
        }

        private int denom;

        public int Denom
        {
            get { return denom; }
            set { denom = value; }
        }

        public override string ToString()
        {
            return string.Format("{0}/{1}", this.number, this.denom);
        }

        public RationalNumber(int numerator = 0, int denominator = 1)
        {
            this.Number = numerator;
            this.Denom = denominator;
        }
    }

    public class Program
    {
        static void Main(string[] args)
        {
            var input = Console.ReadLine().Split().Select(int.Parse).ToArray();

            List<RationalNumber> rationals = new List<RationalNumber>();
        }
    }
}
```




```
for (int i = 0; i < input.Length; i += 2)
{
    rationals.Add
    (
        new RationalNumber(input[i], input[i + 1])
    );
}

Console.WriteLine(string.Join(Environment.NewLine, rationals));
}
```

Задача 2.5. Несъкратима гроб *

Дефинирайте клас `RationalNumber` с `private` полета `numerator` (числител) и `denominator` (знаменател). Въведете от клавиатурата числа на един ред, разделени с интервал които да бъдат числител и знаменатели на гробовете (както в предишния пример) и ги запишете в списък от такива `RationalNumber` обекти. Създайте нов списък, в който са преобразувани така въведените гроби в несъкратими и изведете новия списък.

Подсказки

Създайте метод `int BiggestDivider(int numerator, int denominator)`, който намира Най-големия общ делител на числителя и знаменателя НОД(числител и знаменател). Напишете конструктор, който приема числител и знаменател, извиква в себе си метода `int BiggestDivider(int numerator, int denominator)` и записва в новия списък съкратената гроб по следното правило:

`nod = int BiggestDivider(int numerator, int denominator) => {numerator/nod} {denominator/nod}`. Ползвайте известни алгоритми за намиране на НОД (на Евклид и други)

Примери

Вход	Изход
3 4 3 6 25 100	3/4; 1/2; 1/4

Бонус

1. Изведете новия списък, подреден във възходящ ред
2. Ако не се спазва ограничението за отрицателни знаменатели, вижте дали се налага модификация на метода за подреждане във възходящ или низходящ ред
3. Да се модифицира така, че посоката на подреждане да се въвежда от клавиатурата
4. Задачата да се реши с функционално програмиране

Решение

```
namespace BiggestDivider
{
    public class RationalNumber
    {
```



```
private int number;

public int Number
{
    get { return number; }
    set { number = value; }
}

private int denom;

public int Denom
{
    get { return denom; }
    set { denom = value; }
}

public override string ToString()
{
    return string.Format("{0}/{1}", this.number, this.denom);
}

private int BiggestDivider(int numerator, int denominator)
{
    return denominator == 0 ? numerator : BiggestDivider(denominator,
numerator % denominator);
}

public RationalNumber(int numerator = 0, int denominator = 1)
{
    int gcd = BiggestDivider(numerator, denominator);
    this.Number = numerator / gcd;
    this.Denom = denominator / gcd;
}

public class Program
{
    static void Main(string[] args)
    {
        var input = Console.ReadLine().Split().Select(int.Parse).ToArray();

        List<RationalNumber> rationals = new List<RationalNumber>();

        for (int i = 0; i < input.Length; i += 2)
        {
            rationals.Add
            (
                new RationalNumber(input[i], input[i + 1])
            );
        }

        Console.WriteLine(string.Join(Environment.NewLine, rationals));
    }
}
```



Задача 2.6. Дефиниране на клас Четно число *

Дефинирайте клас EvenNumber с private поле num(числител). Който да ползвате в следната задача:

От клавиатурата на един ред се въвеждат няколко числа и само четните се извеждат. За целта да се ползва списък от числа, в които да се запомнят всички въведени числа. Тези, които са четни от тях да се запаметят в клас EvenNumber и да се изведат на един ред, с разделител запетая.

Примери

Вход	Изход
3 4 5 6 7 8	4, 6, 8

Задача 2.7. Дефиниране на клас Четно число **

Задачата да се реши като се ползва функционално програмиране и ламбда израз за проверка на честността на числата.

Решение

```
namespace EvenNumber
{
    public interface IOddEven
    {
        bool IsOdd(int number);
        bool IsEven(int number);
    }

    public class OddEven : IOddEven
    {
        public bool IsOdd(int number)
        {
            return number % 2 != 0 ? true : false;
        }

        public bool IsEven(int number)
        {
            return number % 2 == 0 ? true : false;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            OddEven checker = new OddEven();

            List<int> numbers = new List<int>();
            numbers = Console.ReadLine().Split().Select(int.Parse).ToList();

            foreach (var number in numbers)
            {
                if (checker.IsEven(number))
                {

```



```
        Console.WriteLine("{0} => Even", number);  
    }  
}  
}
```

Задача 2.8. Дефиниране на клас Нечетно число

Решете предишната задача, с условие, че извеждате нечетни числа.

Решение

```
namespace OddNumber  
{  
    public interface IOddEven  
    {  
        bool IsOdd(int number);  
        bool IsEven(int number);  
    }  
  
    public class OddEven : IOddEven  
    {  
        public bool IsOdd(int number)  
        {  
            return number % 2 != 0 ? true : false;  
        }  
  
        public bool IsEven(int number)  
        {  
            return number % 2 == 0 ? true : false;  
        }  
    }  
  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            OddEven checker = new OddEven();  
  
            List<int> numbers = new List<int>();  
            numbers = Console.ReadLine().Split().Select(int.Parse).ToList();  
  
            foreach (var number in numbers)  
            {  
                if (checker.IsOdd(number))  
                {  
                    Console.WriteLine("{0} => Odd", number);  
                }  
            }  
        }  
    }  
}
```

Задача 2.9. Дефиниране на клас Кратно на „к“ число

Решете задача 5, с условие, че извеждате числата, кратни на число к, което се въвежда от клавиатурата



- На първи ред се въвежда последователност от числа, разделени с интервал
- На втори ред се въвежда число k – на което да са кратни
- На конзолата се извеждат кратните на k , разделени със запетая

Примери

Вход	Изход	Вход	Изход
3 4 5 6 7 8 9 15 28 3	3,6,9,15	3 4 5 6 7 8 9 15 28 4	4,8,28

Тема 3. Шаблонни класове

Задача 3.1. Кутия с Т

Създайте клас `Box<T>`, който може да съхранява всичко.

Той трябва да има два публични метода:

- `void Add(element)`
- `element Remove()`
- `int Count { get; }`

Добавянето трябва да добавя новото най-отгоре. Премахването да взима най-горния.

Примери

```
public static void Main(string[] args)
{
    Box<int> box = new Box<int>();
    box.Add(1);
    box.Add(2);
    box.Add(3);
    Console.WriteLine(box.Remove());
    box.Add(4);
    box.Add(5);
    Console.WriteLine(box.Remove());
}
```

Подсказки

Използвайте формата `Box<T>`, за да създадете шаблонен клас.

Решение

```
namespace BoxT
{
    public class Box<T>
    {
        private List<T> data;

        public Box()
```



```
{
    this.data = new List<T>();
}

public int Count => this.data.Count;

public void Add(T item)
{
    this.data.Add(item);
}

public T Remove()
{
    T temp = this.data.Last();
    this.data.RemoveAt(this.data.Count - 1);
    return temp;
}

internal class Program
{
    static void Main(string[] args)
    {
        Box<int> box = new Box<int>();
        box.Add(1);
        box.Add(2);
        box.Add(3);
        Console.WriteLine(box.Remove());
        box.Add(4);
        box.Add(5);
        Console.WriteLine(box.Remove());
    }
}
```

Задача 3.2. Кутия за всичко

Създайте шаблонен клас Box, който може да бъде инициализиран с произволен тип и да съхранява стойността. Предефинирайте метода ToString() да отпечата типа и стойността на съхраняваните данни във формат {class full name: value}.

Бележка

Класът се използва в следващите задачи. За да вземете пълното име на класа, използвайте свойството [.GetType\(\).FullName](#).

Примери

Вход	Изход
123123	System.Int32: 123123
life in a box	System.String: life in a box

Решение

`namespace TheBox`



```
{
    public class Box<T>
    {
        public int Count { get; private set; }

        private List<T> items;

        public Box()
        {
            items = new List<T>();
            Count = 0;
        }

        public void Add(T item)
        {
            items.Add(item);
            Count++;
        }

        public T Remove()
        {
            T temp = items.Last();
            items.Remove(items.Last());
            Count--;
            return temp;
        }

        public override string ToString()
        {
            string temp = null, type = null;
            foreach (T item in items)
            {
                type = item.GetType().ToString();
                temp += item.ToString();
            }
            return $"{type}: {temp}";
        }

        public void Print()
        {
            foreach (var item in items)
            {
                Console.WriteLine($"{item.GetType()}: {item}");
            }
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            // Integer Box
            Box<int> box1 = new Box<int>();
            box1.Add(123);
            box1.Print();
        }
    }
}
```



```
// String Box
Box<string> box2 = new Box<string>();
box2.Add("hello box");
box2.Print();

// Floating point Box
Box<double> box3 = new Box<double>();
box3.Add(3.14);
box3.Print();
}
}
```

Задача 3.3. Универсална кутия за низове

Използвайте класа, създаден в предната задача и го тествайте с класа System.String. На първия ред ще получите n - броят на низовете, които да прочетете от конзолата. На следващите n реда ще са самите низове. За всеки от тях създайте кутия и извикайте нейния метод ToString(), за да отпечатате данните и на конзолата.

Примери

Вход	Изход
2 life in a box box in a life	System.String: life in a box System.String: box in a life

Решение

```
namespace StringBox
{
    public class Box<T>
    {
        public int Count { get; private set; }

        private List<T> items;

        public Box()
        {
            items = new List<T>();
            Count = 0;
        }

        public void Add(T item)
        {
            items.Add(item);
            Count++;
        }

        public T Remove()
        {
            T temp = items.Last();
            items.Remove(items.Last());
            Count--;
        }
    }
}
```




```
        return temp;
    }

    public override string ToString()
    {
        string temp = null, type = null;
        foreach (T item in items)
        {
            type = item.GetType().ToString();
            temp += item.ToString();
        }
        return $"{type}: {temp}";
    }

    public void Print()
    {
        foreach (var item in items)
        {
            Console.WriteLine($"{item.GetType(): {item}");
        }
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Box<string> box = new Box<string>();
        int n = int.Parse(Console.ReadLine());
        while (n > 0)
        {
            box.Add(Console.ReadLine());
            n--;
        }
        box.Print();
    }
}
```

Задача 3.4. Универсална кутия за цели числа

Използвайте описанието на предната задача, но този път тествайте вашата универсална кутия с цели числа.

Примери

Вход	Изход
3	System.Int32: 7
7	System.Int32: 123
123	System.Int32: 42
42	

Решение

```
namespace IntegerBox
{
```



```
public class Box<T>
{
    public int Count { get; private set; }

    private List<T> items;

    public Box()
    {
        items = new List<T>();
        Count = 0;
    }

    public void Add(T item)
    {
        items.Add(item);
        Count++;
    }

    public T Remove()
    {
        T temp = items.Last();
        items.Remove(items.Last());
        Count--;
        return temp;
    }

    public override string ToString()
    {
        string temp = null, type = null;
        foreach (T item in items)
        {
            type = item.GetType().ToString();
            temp += item.ToString();
        }
        return $"{type}: {temp}";
    }

    public void Print()
    {
        foreach (var item in items)
        {
            Console.WriteLine($"{item.GetType(): {item}");
        }
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Box<int> box = new Box<int>();
        int n = int.Parse(Console.ReadLine());
        while (n > 0)
        {
            box.Add(int.Parse(Console.ReadLine()));
        }
    }
}
```



```
        n--;  
    }  
    box.Print();  
}  
}
```

Задача 3.5. Създател на шаблонен масив

Създайте клас **ArrayCreator** с метод и едно-единствено предефиниране:

- `static T[] Create(int length, T item)`

Методът трябва да връща масив с указаната дължина, в който всеки елемент е установен на предварително зададена стойност.

Примери

```
static void Main(string[] args)  
{  
    string[] strings = ArrayCreator.Create(5, "Pesho");  
    int[] integers = ArrayCreator.Create(10, 33);  
}
```

Решение

```
namespace ArrayCreator
```

```
{  
    public static class ArrayCreator  
    {  
        public static T[] Create<T>(int length, T item)  
        {  
            T[] array = new T[length];  
  
            for (int i = 0; i < length; i++)  
            {  
                array[i] = item;  
            }  
  
            return array;  
        }  
    }  
  
    internal class Program  
    {  
        static void Main(string[] args)  
        {  
            // Array of Strings  
            string[] strings = ArrayCreator.Create(5, "Pesho");  
            Console.WriteLine(string.Join(", ", strings));  
  
            // Array of Integers  
            int[] integers = ArrayCreator.Create(10, 42);  
            Console.WriteLine(string.Join(", ", integers));  
        }  
    }  
}
```



Задача 3.6. Шаблонен метод за размяна на низове

Създайте шаблонен метод, който получава списък с данни от произволен тип и разменя местата на елементите на две указани позиции.

Както в предните примери, прочетете *n* на брой кутии от тип `String` и ги добавете в списък. Тук обаче на следващия ред ще получите команда за размяна, състояща се от два индекса. Използвайте метода, който създадохте, за да размените елементите с позиция, съответстваща на подадените индекси и накрая отпечатайте всички елементи в списъка.

Примери

Вход	Изход
3 Pesho Gosho Swap me with Pesho 0 2	System.String: Swap me with Pesho System.String: Gosho System.String: Pesho

Решение

```
namespace StringSwap
{
    public class StringSwap<T>
    {
        private T[] array;
        private int index;

        public StringSwap(int capacity)
        {
            array = new T[capacity];
            index = 0;
        }

        public void Add(T item)
        {
            array[index] = item;
            index++;
        }

        public void Swap(int first, int second)
        {
            T temp = array[first];
            array[first] = array[second];
            array[second] = temp;
        }

        public void Print()
        {
            foreach (var item in array)
            {
                Console.WriteLine($"{item.GetType()}: {item}");
            }
        }
    }
}
```



```
    }  
  }  
}  
  
internal class Program  
{  
    static void Main(string[] args)  
    {  
        int n = int.Parse(Console.ReadLine());  
        StringSwap<string> strings = new StringSwap<string>(n);  
        for (int i = 0; i < n; i++)  
        {  
            strings.Add(Console.ReadLine());  
        }  
  
        var pos = Console.ReadLine().Split().ToArray();  
        int first = int.Parse(pos[0]);  
        int second = int.Parse(pos[1]);  
  
        strings.Swap(first, second);  
        strings.Print();  
    }  
}
```

Задача 3.7. Шаблонен метод за размяна на цели числа

Използвайте описанието на условието от предната задача, но този път мествайте вашия списък с универсални кутии с цели числа.

Примери

Вход	Изход
3	System.Int32: 42
7	System.Int32: 123
123	System.Int32: 7
42	
0 2	

Решение

```
namespace IntegerSwap  
{  
    public class IntegerSwap<T>  
    {  
        private T[] array;  
        private int index;  
  
        public IntegerSwap(int capacity)  
        {  
            array = new T[capacity];  
            index = 0;  
        }  
  
        public void Add(T item)  
        {
```



```
        array[index] = item;
        index++;
    }

    public void Swap(int first, int second)
    {
        T temp = array[first];
        array[first] = array[second];
        array[second] = temp;
    }

    public void Print()
    {
        foreach (var item in array)
        {
            Console.WriteLine($"{item.GetType()}: {item}");
        }
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        IntegerSwap<int> strings = new IntegerSwap<int>(n);
        for (int i = 0; i < n; i++)
        {
            strings.Add(int.Parse(Console.ReadLine()));
        }

        var pos = Console.ReadLine().Split().ToArray();
        int first = int.Parse(pos[0]);
        int second = int.Parse(pos[1]);

        strings.Swap(first, second);
        strings.Print();
    }
}
```

Задача 3.8. Шаблонен метод за броене на низове

Създайте метод който получава като параметър списък от кой да е от типовете данни, които могат да бъдат сравнявани и един елемент от същия тип. Методът трябва да връща броя на елементите, които са по-големи по стойност от подадения елемент. Променете вашия клас Box така, че да поддържа сравняване на стойностите на съхранените данни.

На първия ред ще получите n - броят на елементите, които да добавите в списъка. На следващите n реда ще получите самите елементи. На последния ред ще е стойността на елемента, спрямо който ще сравнявате всеки един елемент от списъка.



Примери

Вход	Изход
3 aa aaa bb aa	2

Решение

namespace StringCounter

```
{  
    public class Box<T> where T : IComparable  
    {  
        private List<T> items;  
  
        public int Count { get; private set; }  
  
        public Box()  
        {  
            items = new List<T>();  
            Count = 0;  
        }  
  
        public void Add(T item)  
        {  
            items.Add(item);  
            Count++;  
        }  
  
        public T Remove()  
        {  
            T temp = items.Last();  
            items.Remove(items.Last());  
            Count--;  
            return temp;  
        }  
  
        public override string ToString()  
        {  
            string temp = null, type = null;  
            foreach (T item in items)  
            {  
                type = item.GetType().ToString();  
                temp += item.ToString();  
            }  
            return $"{type}: {temp}";  
        }  
  
        public void Print()  
        {  
            foreach (var item in items)  
            {  
                Console.WriteLine($"{item.GetType()}: {item}");  
            }  
        }  
    }  
}
```



```
    }  
}  
  
public int BiggerThan(T element)  
{  
    int count = 0;  
    foreach (var item in items)  
    {  
        if (item.CompareTo(element) > 0)  
        {  
            count++;  
        }  
    }  
    return count;  
}  
}  
  
internal class Program  
{  
    static void Main(string[] args)  
    {  
        Box<string> box = new Box<string>();  
  
        int n = int.Parse(Console.ReadLine());  
        for (int i = 0; i < n; i++)  
        {  
            box.Add(Console.ReadLine());  
        }  
        string element = Console.ReadLine();  
  
        Console.WriteLine(box.BiggerThan(element));  
    }  
}
```

Задача 3.9. Шаблонен метод за броене на дробни числа

Използвайте описанието на условието от предната задача, но този път тествайте вашия списък с универсални кутии с числа от тип double.

Примери

Вход	Изход
3 7.13 123.22 42.78 7.55	2

Решение

```
namespace FloatCounter  
{  
    public class Box<T> where T : IComparable  
    {  
        private List<T> items;
```




```
public int Count { get; private set; }

public Box()
{
    items = new List<T>();
    Count = 0;
}

public void Add(T item)
{
    items.Add(item);
    Count++;
}

public T Remove()
{
    T temp = items.Last();
    items.Remove(items.Last());
    Count--;
    return temp;
}

public override string ToString()
{
    string temp = null, type = null;
    foreach (T item in items)
    {
        type = item.GetType().ToString();
        temp += item.ToString();
    }
    return $"{type}: {temp}";
}

public void Print()
{
    foreach (var item in items)
    {
        Console.WriteLine($"{item.GetType()}: {item}");
    }
}

public int BiggerThan(T element)
{
    int count = 0;
    foreach (var item in items)
    {
        if (item.CompareTo(element) > 0)
        {
            count++;
        }
    }
    return count;
}
}
```



```
internal class Program
{
    static void Main(string[] args)
    {
        Box<double> box = new Box<double>();

        int n = int.Parse(Console.ReadLine());
        for (int i = 0; i < n; i++)
        {
            box.Add(double.Parse(Console.ReadLine()));
        }
        double element = double.Parse(Console.ReadLine());

        Console.WriteLine(box.BiggerThan(element));
    }
}
```

Задача 3.10. Универсална везна

Създайте клас `Scale<T>`, Съдържа два елемента: `left` и `right`. Получава ги чрез своя единствен конструктор:

- `Scale(T left, T right)`

Везната трябва да има един-единствен метод:

- `T getHeavier()`

По-големият от двата елемента е по-тежък. Методът трябва да връща `default(T)`, ако елементите са еднакви.

Решение

```
namespace Scale
{
    public class Scale<T> where T : IComparable<T>
    {
        private T left;
        private T right;

        public Scale(T left, T right)
        {
            this.left = left;
            this.right = right;
        }

        public T GetHavier()
        {
            if (left.CompareTo(right) > 0) return left;
            else if (left.CompareTo(right) < 0) return right;
            return default(T);
        }
    }
}

internal class Program
```



```
{
    static void Main(string[] args)
    {
        // Integer
        Scale<int> intScale = new Scale<int>(32, 41);
        Console.WriteLine(intScale.GetHavier());

        // String
        Scale<string> stringScale = new Scale<string>("Ivan", "Peter");
        Console.WriteLine(stringScale.GetHavier());

        // Boolean
        Scale<bool> boolScale = new Scale<bool>(true, false);
        Console.WriteLine(boolScale.GetHavier());

        // Double
        Scale<double> doubleScale = new Scale<double>(2.34, 3.45);
        Console.WriteLine(doubleScale.GetHavier());
    }
}
```

Задача 3.11. Подобрен списък

Създайте универсална структура данни, която може да съхранява произволен тип данни, който може да бъде сравняван. Реализирайте функциите:

- void Add(T element)
- T Remove(int index)
- bool Contains(T element)
- void Swap(int index1, int index2)
- int CountGreaterThan(T element)
- T Max()
- T Min()

Създайте команден интерпретатор, който чете команди и променя подобрения списък, който сте създали. Инициализирайте списъка да съхранява низове. Реализирайте командите:

- Add <element> - добавя даден елемент в края на списъка
- Remove <index> - премахва елемента, намиращ се на указаната позиция
- Contains <element> - отпечатва дали списъкът съдържа даден елемент (True или False)
- Swap <index> <index> - разменя местата на елементите с указаните индекси
- Greater <element> - преброява елементите, които са по-големи от подадения елемент и отпечатва техния брой
- Max - отпечатва максималния елемент от списъка
- Min - отпечатва минималния елемент от списъка
- Print - отпечатва всички елементи в списъка, всеки на отделен ред
- END - приключва с четенето на командите

Няма да има никакви невалидни команди във входните данни.



Примери

Вход	Изход
Add aa	cc
Add bb	aa
Add cc	2
Max	True
Min	cc
Greater aa	bb
Swap 0 2	aa
Contains aa	
Print	
END	

Решение

```
namespace CustomList
{
    public static class Console
    {
        public static void WriteLine<T>(T element)
        {
            var originalColor = System.Console.ForegroundColor;
            System.Console.ForegroundColor = ConsoleColor.Green;
            System.Console.WriteLine(element);
            System.Console.ForegroundColor = originalColor;
        }

        public static string ReadLine()
        {
            return System.Console.ReadLine();
        }
    }

    public class CustomList<T> where T : IComparable
    {
        private List<T> list;

        public int Count { get; private set; }

        public CustomList()
        {
            this.list = new List<T>();
            this.Count = 0;
        }

        public void Add(T element)
        {
            this.list.Add(element);
            this.Count++;
        }

        public T Remove(int index)
        {
        }
```



```
        var element = this.list[index];
        this.list.Remove(element);
        this.Count--;
        return element;
    }
    public bool Contains(T element)
    {
        return this.list.Contains(element);
    }

    public void Swap(int index1, int index2)
    {
        var temp = this.list[index1];
        this.list[index1] = this.list[index2];
        this.list[index2] = temp;
    }

    public int CountGreaterThan(T element)
    {
        return this.list.Where(x => x.CompareTo(element) > 0).Count();
    }
    public T Max()
    {
        return this.list.Max();
    }

    public T Min()
    {
        return this.list.Min();
    }

    public void Print()
    {
        foreach (var item in this.list)
        {
            Console.WriteLine(item);
        }
    }

    public override string ToString()
    {
        string temp = null;
        foreach (var it in this.list)
        {
            temp += $"{it} ";
        }
        return temp;
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        CustomList<string> list = new CustomList<string>();
```



```
var cmd = Console.ReadLine().Split().ToArray();
while (cmd[0] != "END")
{
    switch (cmd[0])
    {
        case "Add": list.Add(cmd[1]); break;
        case "Remove":
            Console.WriteLine(list.Remove(int.Parse(cmd[1])));
            break;
        case "Contains":
            Console.WriteLine(list.Contains(cmd[1]));
            break;
        case "Swap":
            list.Swap(int.Parse(cmd[1]), int.Parse(cmd[2]));
            break;
        case "Greater":
            Console.WriteLine(list.CountGreaterThan(cmd[1]));
            break;
        case "Max":
            Console.WriteLine(list.Max());
            break;
        case "Min":
            Console.WriteLine(list.Min());
            break;
        case "Print":
            list.Print();
            break;
    }
    cmd = Console.ReadLine().Split().ToArray();
}
}
```

Задача 3.12. Сортиране на подобрения списък

Разширете решението на предната задача чрез създаване на допълнителен клас `Sorter`. Той трябва да има един-единствен статичен метод `Sort()`, който може да сортира обекти от `mon CustomList`, съдържащи данни от произволен `тип`, който подлежи на сравняване. Разширете списъка с команди, така че да поддържа една допълнителна команда `Sort`:

- `Sort` - сортира елементите в списъка в нарастващ ред.

Примери

Вход	Изход
Add cc Add bb Add aa Sort Print	aa bb cc



END	
-----	--

Решение

namespace CustomListSort

```
{
    public static class Console
    {
        public static void WriteLine<T>(T element)
        {
            var originalColor = System.Console.ForegroundColor;
            System.Console.ForegroundColor = ConsoleColor.Green;
            System.Console.WriteLine(element);
            System.Console.ForegroundColor = originalColor;
        }

        public static string ReadLine()
        {
            return System.Console.ReadLine();
        }
    }

    public class CustomList<T> where T : IComparable
    {
        private List<T> list;

        public int Count { get; private set; }

        public CustomList()
        {
            this.list = new List<T>();
            this.Count = 0;
        }

        public void Add(T element)
        {
            this.list.Add(element);
            this.Count++;
        }

        public T Remove(int index)
        {
            var element = this.list[index];
            this.list.Remove(element);
            this.Count--;
            return element;
        }

        public bool Contains(T element)
        {
            return this.list.Contains(element);
        }

        public void Swap(int index1, int index2)
        {
            var temp = this.list[index1];
```



```
        this.list[index1] = this.list[index2];
        this.list[index2] = temp;
    }

    public int CountGreaterThan(T element)
    {
        return this.list.Where(x => x.CompareTo(element) > 0).Count();
    }

    public T Max()
    {
        return this.list.Max();
    }

    public T Min()
    {
        return this.list.Min();
    }

    public void Print()
    {
        foreach (var item in this.list)
        {
            Console.WriteLine(item);
        }
    }

    public void Sort()
    {
        this.list = this.list.OrderBy(a => a).ToList();
    }

    public override string ToString()
    {
        string temp = null;
        foreach (var it in this.list)
        {
            temp += $"{it} ";
        }
        return temp;
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        CustomList<string> list = new CustomList<string>();

        var cmd = Console.ReadLine().Split().ToArray();
        while (cmd[0] != "END")
        {
            switch (cmd[0])
            {
                case "Add": list.Add(cmd[1]); break;
            }
        }
    }
}
```




```
        case "Remove":  
            Console.WriteLine(list.Remove(int.Parse(cmd[1])));  
            break;  
        case "Contains":  
            Console.WriteLine(list.Contains(cmd[1]));  
            break;  
        case "Swap":  
            list.Swap(int.Parse(cmd[1]), int.Parse(cmd[2]));  
            break;  
        case "Greater":  
            Console.WriteLine(list.CountGreaterThan(cmd[1]));  
            break;  
        case "Max":  
            Console.WriteLine(list.Max());  
            break;  
        case "Min":  
            Console.WriteLine(list.Min());  
            break;  
        case "Print":  
            list.Print();  
            break;  
        case "Sort":  
            list.Sort();  
            break;  
    }  
    cmd = Console.ReadLine().Split().ToArray();  
}  
}  
}
```

Задача 3.13. *Обхождане на подобрения списък

За всяка от командите за отпечатване вероятно сте използвали цикъл for. Разширете вашия подобрен списък, така че да реализира интерфейса IEnumerable<T>. Това би позволило да обхождате вашия списък с командата foreach.

Примери

Вход	Изход
Add aa	cc
Add bb	aa
Add cc	2
Max	cc
Min	bb
Greater aa	aa
Swap 0 2	
Print	
END	

Решение

```
using System.Collections;
```



```
namespace CustomListEnumerable
{
    public static class Console
    {
        public static void WriteLine<T>(T element)
        {
            var originalColor = System.Console.ForegroundColor;
            System.Console.ForegroundColor = ConsoleColor.Green;
            System.Console.WriteLine(element);
            System.Console.ForegroundColor = originalColor;
        }

        public static string ReadLine()
        {
            return System.Console.ReadLine();
        }
    }

    public class CustomList<T> where T : IComparable, IEnumerable
    {
        private List<T> list;

        public int Count { get; private set; }

        public CustomList()
        {
            this.list = new List<T>();
            this.Count = 0;
        }

        public void Add(T element)
        {
            this.list.Add(element);
            this.Count++;
        }

        public T Remove(int index)
        {
            var element = this.list[index];
            this.list.Remove(element);
            this.Count--;
            return element;
        }

        public bool Contains(T element)
        {
            return this.list.Contains(element);
        }

        public void Swap(int index1, int index2)
        {
            var temp = this.list[index1];
            this.list[index1] = this.list[index2];
            this.list[index2] = temp;
        }
    }
}
```



```
}

public int CountGreaterThan(T element)
{
    return this.list.Where(x => x.CompareTo(element) > 0).Count();
}

public T Max()
{
    return this.list.Max();
}

public T Min()
{
    return this.list.Min();
}

public void Print()
{
    foreach (var item in this.list)
    {
        Console.WriteLine(item);
    }
}

public void Sort()
{
    this.list = this.list.OrderBy(a => a).ToList();
}

public override string ToString()
{
    string temp = null;
    foreach (var it in this.list)
    {
        temp += $"{it} ";
    }
    return temp;
}
}

internal class Program
{
    static void Main(string[] args)
    {
        CustomList<string> list = new CustomList<string>();

        var cmd = Console.ReadLine().Split().ToArray();
        while (cmd[0] != "END")
        {
            switch (cmd[0])
            {
                case "Add": list.Add(cmd[1]); break;
                case "Remove":
                    Console.WriteLine(list.Remove(int.Parse(cmd[1])));
            }
        }
    }
}
```



```
        break;
    case "Contains":
        Console.WriteLine(list.Contains(cmd[1]));
        break;
    case "Swap":
        list.Swap(int.Parse(cmd[1]), int.Parse(cmd[2]));
        break;
    case "Greater":
        Console.WriteLine(list.CountGreaterThan(cmd[1]));
        break;
    case "Max":
        Console.WriteLine(list.Max());
        break;
    case "Min":
        Console.WriteLine(list.Min());
        break;
    case "Print":
        list.Print();
        break;
    case "Sort":
        list.Sort();
        break;
    }
    cmd = Console.ReadLine().Split().ToArray();
}
}
}
```

Задача 3.14. Tuple

Има нещо много странно в C#. Нарича се [Tuple](#). Това е клас, който може да съхранява няколко обекта, но нека се фокусираме върху `tuple` Tuple, който съхранява два обекта. Първият е "item1", а вторият - "item2". Това е нещо подобно на `KeyValuePair` с изключение на това, че просто съхранява елементи, които не са нито ключове, нито стойности. Странността идва от факта, че нямате никаква идея какво съдържат тези елементи. Името на класа нищо не ви подсказва, методите които има - също. И така, нека си представим, че по някаква причина бихме искали да се опитаме сами да направим такъв клас, ей така - просто за да упражним шаблоните.

Задачата: създайте клас "Tuple", съдържащ два обекта. Както споменахме, първият ще е "item1", а вторият - "item2". Тънкостта тук идва от това, да накараме класа да поддържа шаблони. Това ще рече когато създаваме нов обект от клас "Tuple", трябва да начин изрично да укажем типа и на двата елемента поотделно.

Вход

Входните данни включват три реда:

- Първият съдържа името на човек и адресът му. Те са отделени с интервал(и). Вашата задача е да ги прочетете в tuple-а и да ги отпечатате на конзолата. Форматът на входните данни е:



- <<first name> <last name>> <address>
- Вторият ред съдържа име на човек и количеството бира (int), което той може да изпие. Формат:
<name> <liters of beer>
 - Последният ред съдържа Integer и Double. Формат:
<Integer> <Double>

Изход

- Отпечатайте елементите на tuple-а във формат: {item1} -> {item2}

Ограничения

Използвайте добрите практики, които сме учили. Създайте клас и му добавете getters и setters за клас-променливите му. Входните данни ще са валидни, няма нужда изрично да ги проверяваме!

Пример

Вход	Изход
Sofka Tripova Stolipinovo	Sofka Tripova -> Stolipinovo
Az 2	Az -> 2
23 21.23212321	23 -> 21.23212321

Решение

namespace Tuple

```
{  
    public static class Console  
    {  
        public static void WriteLine<T>(T element)  
        {  
            var originalColor = System.Console.ForegroundColor;  
            System.Console.ForegroundColor = ConsoleColor.Green;  
            System.Console.WriteLine(element);  
            System.Console.ForegroundColor = originalColor;  
        }  
  
        public static string ReadLine()  
        {  
            return System.Console.ReadLine();  
        }  
    }  
  
    public class Tuple<T, U>  
    {  
        private T item1;  
  
        public T Item1  
        {  
            get { return item1; }  
            set { item1 = value; }  
        }  
  
        private U item2;  
  
        public U Item2  
        {  

```



```
        get { return item2; }
        set { item2 = value; }
    }

    public Tuple(T item1 = default(T), U item2 = default(U))
    {
        this.Item1 = item1;
        this.Item2 = item2;
    }

    public override string ToString()
    {
        return $"{this.Item1} -> {this.Item2}";
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        string[] firstLine = Console.ReadLine().Split(' ').ToArray();
        string[] secondLine = Console.ReadLine().Split(' ').ToArray();
        string[] thirdLine = Console.ReadLine().Split(' ').ToArray();

        Tuple<string, string> nameAddress = new Tuple<string,
string>(firstLine[0] + " " + firstLine[1], firstLine[2]);
        Tuple<string, int> nameBeer = new Tuple<string, int>(secondLine[0],
int.Parse(secondLine[1]));
        Tuple<int, double> intDouble = new Tuple<int,
double>(int.Parse(thirdLine[0]), double.Parse(thirdLine[1]));

        Console.WriteLine(nameAddress);
        Console.WriteLine(nameBeer);
        Console.WriteLine(intDouble);
    }
}
```

Задача 3.15. Threeuple

Създайте клас Threeuple. Its name is telling us, that it will hold no longer, just a pair of objects. The task is simple, our Threeuple should hold three objects. Make it have getters and setters. You can even extend the previous class

Вход

Входните данни се състоят от три реда:

- Първият ред съдържа име, адрес и град, във формат:
<<first name> <last name>> <address> <town>
- Вторият ред съдържа име, литри бира и булева променлива със стойност drunk или not. Форматът е:
<name> <liters of beer> <drunk or not>
- Третият ред съдържа име, наличност по банковата сметка (double) и име на банката. Форматът е:
<name> <account balance> <bank name>



Изход

- Omneчamaйme Threeuple обектume във формат: {firstElement} -> {secondElement} -> {thirdElement}

Примери

Вход	Изход
Sofka Tripova Stolipinovo Plovdiv MitkoShtaigata 18 drunk SashoKompota 0.10 NkqfaBanka	Sofka Tripova -> Stolipinovo -> Plovdiv MitkoShtaigata -> 18 -> True SashoKompota -> 0.1 -> NkqfaBanka
Ivan Ivanov Tepeto Plovdiv Mitko 18 not Sasho 0.10 NGB	Ivan Ivanov -> Tepeto -> Plovdiv Mitko -> 18 -> False Sasho -> 0.1 -> NGB

Бележки

Може да използвате и наградите решението на предната задача.

Решение

namespace Threeuple

```
{  
    public static class Console  
    {  
        public static void WriteLine<T>(T element)  
        {  
            var originalColor = System.Console.ForegroundColor;  
            System.Console.ForegroundColor = ConsoleColor.Green;  
            System.Console.WriteLine(element);  
            System.Console.ForegroundColor = originalColor;  
        }  
  
        public static string ReadLine()  
        {  
            return System.Console.ReadLine();  
        }  
    }  
  
    public class Threeuple<T, U, V>  
    {  
        private T item1;  
  
        public T Item1  
        {  
            get { return item1; }  
            set { item1 = value; }  
        }  
  
        private U item2;  
  
        public U Item2  
        {  
            get { return item2; }  
            set { item2 = value; }  
        }  
    }  
}
```



```
private V item3;

public V Item3
{
    get { return item3; }
    set { item3 = value; }
}

public Triple<T item1 = default(T), U item2 = default(U), V item3 =
default(V)>
{
    this.Item1 = item1;
    this.Item2 = item2;
    this.Item3 = item3;
}

public override string ToString()
{
    return $"{this.Item1} -> {this.Item2} -> {this.Item3}";
}
}

internal class Program
{
    static void Main(string[] args)
    {
        string[] firstLine = Console.ReadLine().Split(' ').ToArray();
        string[] secondLine = Console.ReadLine().Split(' ').ToArray();
        string[] thirdLine = Console.ReadLine().Split(' ').ToArray();

        bool isDrunk = secondLine[2] == "drunk" ? true : false;
        Triple<string, string, string> nameAddressTown = new
Triple<string, string, string>
        (firstLine[0] + " " + firstLine[1], firstLine[2], firstLine[3]);

        Triple<string, int, bool> nameBeerDrunk = new Triple<string, int,
bool>
        (secondLine[0], int.Parse(secondLine[1]), isDrunk);

        Triple<string, double, string> nameBalanceBank = new
Triple<string, double, string>
        (thirdLine[0], double.Parse(thirdLine[1]), thirdLine[2]);

        Console.WriteLine(nameAddressTown);
        Console.WriteLine(nameBeerDrunk);
        Console.WriteLine(nameBalanceBank);
    }
}
```




Тема 4. Наследяване, абстракция, интерфейси

Задача 4.1. Единично наследяване

Създайте два класа `Animal` и `Dog`.

`Animal` с единствен публичен метод `Eat()`, който отпечатава: "eating...".

`Dog` с единствен публичен метод `Bark()`, който отпечатава: "barking...".

`Dog` трябва да наследява `Animal`.

```
Dog dog = new Dog();  
dog.Eat();  
dog.Bark();
```

Подсказка

Използвайте : оператора, за да построите йерархията

Решение

```
namespace SingleInheritance  
{  
    public class Animal  
    {  
        public void Eat()  
        {  
            Console.WriteLine("Eating...");  
        }  
    }  
  
    public class Dog : Animal  
    {  
        public void Bark()  
        {  
            Console.WriteLine("Barking...");  
        }  
    }  
  
    internal class Program  
    {  
        static void Main(string[] args)  
        {  
            Dog dog = new Dog();  
            dog.Eat();  
            dog.Bark();  
        }  
    }  
}
```

Задача 4.2. Наследяване на много нива

Създайте три класа `Animal`, `Dog` и `Puppy`.

`Animal` с единствен публичен метод `Eat()`, който отпечатава: "eating...".

`Dog` с единствен публичен метод `Bark()`, който отпечатава: "barking...".



Puppy с единствен публичен метод Weep(), който отпечата: "weeping...".

Dog трябва да наследява Animal. Puppy трябва да наследява Dog.

```
Puppy puppy = new Puppy();  
puppy.Eat();  
puppy.Bark();  
puppy.Weep();
```

Решение

```
namespace MultipleInheritance  
{  
    public class Animal  
    {  
        public void Eat()  
        {  
            Console.WriteLine("Eating...");  
        }  
    }  
  
    public class Dog : Animal  
    {  
        public void Bark()  
        {  
            Console.WriteLine("Barking...");  
        }  
    }  
  
    public class Puppy : Dog  
    {  
        public void Weep()  
        {  
            Console.WriteLine("Weeping...");  
        }  
    }  
  
    internal class Program  
    {  
        static void Main(string[] args)  
        {  
            Puppy puppy = new Puppy();  
            puppy.Eat();  
            puppy.Bark();  
            puppy.Weep();  
        }  
    }  
}
```

Задача 4.3. Йерархично наследяване

Създайте три класа Animal, Dog и Cat.

Animal с единствен публичен метод Eat(), който отпечата: "eating...".

Dog с единствен публичен метод Bark(), който отпечата: "barking...".



Cat с единствен публичен метод Meow(), който отпечата: "meowing...".

Dog и Cat трябва да наследят Animal.

```
Dog dog = new Dog();  
dog.Eat();  
dog.Bark();
```

```
Cat cat = new Cat();  
cat.Eat();  
cat.Meow();
```

Решение

```
namespace HierarchyInheritance  
{  
    public class Animal  
    {  
        public void Eat()  
        {  
            Console.WriteLine("Eating...");  
        }  
    }  
  
    public class Dog : Animal  
    {  
        public void Bark()  
        {  
            Console.WriteLine("Barking...");  
        }  
    }  
  
    public class Cat : Animal  
    {  
        public void Meow()  
        {  
            Console.WriteLine("Meowing...");  
        }  
    }  
  
    internal class Program  
    {  
        static void Main(string[] args)  
        {  
            Dog dog = new Dog();  
            dog.Eat();  
            dog.Bark();  
  
            Cat cat = new Cat();  
            cat.Eat();  
            cat.Meow();  
        }  
    }  
}
```



}

Задача 4.4. Случаен списък

Създайте клас `RandomList`, който има всичката функционалност на `ArrayList`.

Добавете допълнителна функционалност, която връща и премахва случаен елемент от списъка.

- Публичен метод: `RandomString(): string`

Решение

```
namespace RandomList
{
    public class RandomList : ArrayList
    {
        private Random random = new Random();

        public object RandomObject()
        {
            var index = random.Next(0, base.Count - 1);
            object element = base[index];
            base.Remove(element);
            return element;
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            RandomList list = new RandomList();
            list.Add(42); // int
            list.Add("fourty two"); // string
            list.Add(3.14); // float
            list.Add(true); // bool
            Console.WriteLine(String.Join(", ", list.ToArray()));
            Console.WriteLine(list.RandomObject());
            Console.WriteLine(list.RandomObject());
        }
    }
}
```

Задача 4.5. Стек от низове

Създайте клас `StackOfStrings`, който може да пази само низове и има следната функционалност:

- Private поле: `data: List<string>`
- Публичен метод: `Push(string item): void`
- Публичен метод: `Pop(): string`
- Публичен метод: `Peek(): string`
- Публичен метод: `IsEmpty(): bool`



Използвайте композиция/делегиране, за да имате поле, в което пази информацията от стека.

Решение

`namespace StackOfStrings`

```
{
    public class StackOfStrings<T> : List<T>, IEnumerable<T>
    {
        private List<T> items = new List<T>();

        public void Push(T item)
        {
            items.Add(item);
        }

        public T Pop()
        {
            if (IsEmpty())
            {
                throw new Exception("Stack is empty!");
            }
            T item = this.items.Last();
            items.Remove(item);
            return item;
        }

        public T Peek()
        {
            return items.Last();
        }

        public bool IsEmpty()
        {
            return items.Count == 0 ? true : false;
        }

        public new IEnumerator<T> GetEnumerator()
        {
            for (int i = 0; i < items.Count; i++)
            {
                yield return items[i];
            }
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }

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



```
static void Main(string[] args)
{
    StackOfStrings<string> stack = new StackOfStrings<string>();
    stack.Add("112");
    stack.Add("911");
    stack.Add("166");

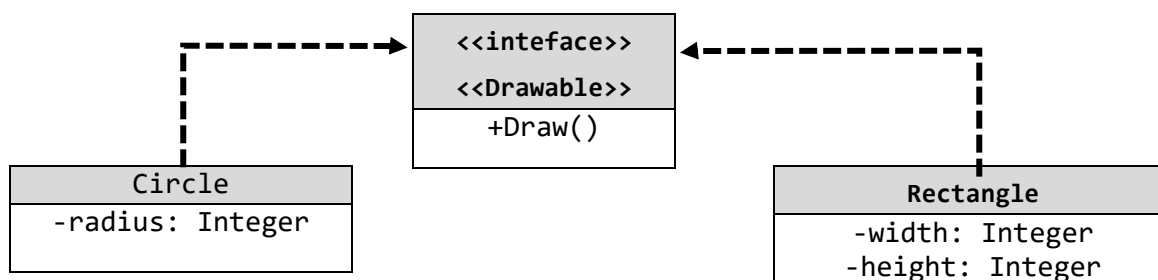
    Console.WriteLine(String.Join(", ", stack));

    while (!stack.IsEmpty())
    {
        Console.Write("{0}, ", stack.Pop());
    }

    try
    {
        Console.WriteLine(stack.Pop());
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
}
```

Задача 4.6. Фигури

Постройте йерархията от интерфейси и класове:



Трябва да може да използвате класовете по сходен начин:

```
Program.cs
```



```
var radius = int.Parse(Console.ReadLine());
IDrawable circle = new Circle(radius);

var width = int.Parse(Console.ReadLine());
var height = int.Parse(Console.ReadLine());
IDrawable rect = new Rectangle(width, height);

circle.Draw();
rect.Draw();
```

Примери

Вход	Изход
3 4 5	***** ** ** ** ** * * ** ** ** ** ***** **** * * * * * * ****

Решение

```
namespace Figures
{
    public interface IDrawable
    {
        public void Draw();
    }

    public class Circle : IDrawable
    {
        private double Radius;

        public Circle(double radius)
        {
            Radius = radius;
        }

        public void Draw()
        {
            double r_in = this.Radius - 0.4;
            double r_out = this.Radius + 0.4;
            for (double y = this.Radius; y >= -this.Radius; --y)
            {
                for (double x = -this.Radius; x < r_out; x += 0.5)
                {
```



```
        double value = x * x + y * y;
        if (value >= r_in * r_in && value <= r_out * r_out)
            Console.WriteLine("*");
        else
            Console.Write(" ");
    }
    Console.WriteLine();
}

}

public class Rectangle : IDrawable
{
    private int Width;
    private int Height;

    public Rectangle(int width, int height)
    {
        Width = width;
        Height = height;
    }

    public void Draw()
    {
        DrawLine(this.Width, '*', '*');
        for (int i = 1; i < this.Height - 1; ++i)
            DrawLine(this.Width, '*', ' ');
        DrawLine(this.Width, '*', '*');
    }

    private void DrawLine(int width, char end, char mid)
    {
        Console.Write(end);
        for (int i = 1; i < width - 1; ++i)
            Console.Write(mid);
        Console.WriteLine(end);
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        // Circle
        var radius = int.Parse(Console.ReadLine());
        IDrawable circle = new Circle(radius);

        // Rectangle
        var width = int.Parse(Console.ReadLine());
        var height = int.Parse(Console.ReadLine());
        IDrawable rect = new Rectangle(width, height);

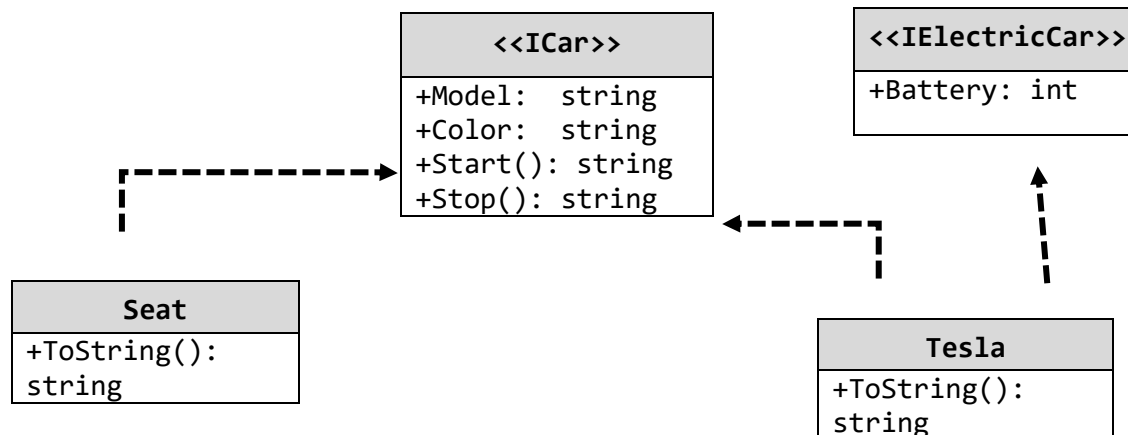
        // Draw
        circle.Draw();
    }
}
```




```
        rect.Draw();  
    }  
}
```

Задача 4.7. Коли

Постройте йерархията от интерфейси и класове:



Вашата йерархия трябва да може да се ползва със следния код:

```
Program.cs

ICar seat = new Seat("Leon", "Grey");
ICar tesla = new Tesla("Model 3", "Red", 2);

Console.WriteLine(seat.ToString());
Console.WriteLine(tesla.ToString());
```

Примери

Input	Output
	Grey Seat Leon Engine start Breaaak! Red Tesla Model 3 with 2 Batteries Engine start Breaaak!

Решение

```
namespace Cars
{
    public interface ICar
    {
        public string Model { get; }
```



```
        public string Color { get; }
        public string Start();
        public string Stop();
    }

    public interface IElectricCar
    {
        public int Battery { get; }
    }

    public class Seat : ICar
    {
        public string Model { get; private set; }

        public string Color { get; private set; }

        public string Start() => "Engine start!";

        public string Stop() => "Breaaaak!";

        public Seat(string model, string color)
        {
            Model = model;
            Color = color;
        }

        public override string ToString()
        {
            return String.Format("{0} Seat {1}", Color, Model);
        }
    }

    public class Tesla : ICar, IElectricCar
    {
        public string Model { get; private set; }

        public string Color { get; private set; }

        public int Battery { get; private set; }

        public string Start() => "Engine start!";

        public string Stop() => "Breaaaak!";

        public Tesla(string model, string color, int battery)
        {
            Model = model;
            Color = color;
            Battery = battery;
        }

        public override string ToString()
        {
            return String.Format("{0} Tesla {1} with {2} Batteries", Color, Model,
Battery);
```



```
    }  
}  
  
internal class Program  
{  
    static void Main(string[] args)  
    {  
        // Seat  
        ICar seat = new Seat("Leon", "Grey");  
        Console.WriteLine(seat.ToString());  
        Console.WriteLine(seat.Start());  
        Console.WriteLine(seat.Stop());  
  
        // Tesla  
        ICar tesla = new Tesla("Model 3", "Red", 2);  
        Console.WriteLine(tesla.ToString());  
        Console.WriteLine(tesla.Start());  
        Console.WriteLine(tesla.Stop());  
    }  
}
```

Задача 4.8. Дефиниране на интерфейс IPerson

Дефинирайте интерфейс IPerson със свойства Name и Age. Дефинирайте клас Citizen, който имплементира IPerson и има конструктор, който взема string с името и int с възрастта.

За да се тества успешно задача, добавете следния код към Main метода си:

```
public static void Main(string[] args)  
{  
    Type personInterface = typeof(Citizen).GetInterface("IPerson");  
    PropertyInfo[] properties = personInterface.GetProperties();  
    Console.WriteLine(properties.Length);  
    string name = Console.ReadLine();  
    int age = int.Parse(Console.ReadLine());  
    IPerson person = new Citizen(name, age);  
    Console.WriteLine(person.Name);  
    Console.WriteLine(person.Age);  
}
```

Ако сте дефинирали интерфейса и сте го имплементирали правилно, местовете би трябвало да минат.

Примери

Вход	Изход
Pesho 25	2 Pesho 25



Решение

```
namespace Person
{
    public interface IPerson
    {
        public string Name { get; }
        public int Age { get; }
    }

    public class Citizen : IPerson
    {
        public string Name { get; private set; }

        public int Age { get; private set; }

        public Citizen(string name, int age)
        {
            Name = name;
            Age = age;
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            Type personInterface = typeof(Citizen).GetInterface("IPerson");
            PropertyInfo[] properties = personInterface.GetProperties();
            Console.WriteLine(properties.Length);
            string name = Console.ReadLine();
            int age = int.Parse(Console.ReadLine());
            IPerson person = new Citizen(name, age);
            Console.WriteLine(person.Name);
            Console.WriteLine(person.Age);
        }
    }
}
```

Задача 4.9. Множествена имплементация

Използвайки кода от предната задача, дефинирайте интерфейс `Identifiable` със свойство `Id` от `mun string` и интерфейс `Birthable` със свойство `Birthdate` от `mun string` и ги имплементирайте в клас `Citizen`. Пренапишете конструктора, така че да приема новите параметри.

За да се тества успешно задача, добавете следния код към `Main` метода си:

```
public static void Main(string[] args)
{
    Type identifiableInterface =
    typeof(Citizen).GetInterface("IIdentifiable");
    Type birthableInterface = typeof(Citizen).GetInterface("IBirthable");
    PropertyInfo[] properties = identifiableInterface.GetProperties();
    Console.WriteLine(properties.Length);
}
```



```
Console.WriteLine(properties[0].PropertyType.Name);
properties = birthableInterface.GetProperties();
Console.WriteLine(properties.Length);
Console.WriteLine(properties[0].PropertyType.Name);
string name = Console.ReadLine();
int age = int.Parse(Console.ReadLine());
string id = Console.ReadLine();
string birthdate = Console.ReadLine();
IIdentifiable identifiable = new Citizen(name, age, id, birthdate);
IBirthable birthable = new Citizen(name, age, id, birthdate);
}
```

Ако сте дефинирали интерфейса и сте го имплементирали правилно, местовите би трябвало да минат.

Примери

Вход	Изход
Pesho	1
25	String
9105152287	1
15/05/1991	String

Решение

```
namespace MultipleInterfaces
{
    public interface IPerson
    {
        public string Name { get; }
        public int Age { get; }
    }

    public interface IIdentifiable
    {
        public string Id { get; }
    }

    public interface IBirthable
    {
        public string Birthdate { get; }
    }

    public class Citizen : IPerson, IBirthable, IIdentifiable
    {
        public string Name { get; private set; }

        public int Age { get; private set; }

        public string Birthdate { get; private set; }

        public string Id { get; private set; }
    }
}
```



```
public Citizen(string name, int age, string id, string birthdate)
{
    Name = name;
    Age = age;
    Id = id;
    Birthdate = birthdate;
}

internal class Program
{
    static void Main(string[] args)
    {
        Type identifiableInterface =
typeof(Citizen).GetInterface("IIdentifiable");
        Type birthableInterface = typeof(Citizen).GetInterface("IBirthable");
        PropertyInfo[] properties = identifiableInterface.GetProperties();
        Console.WriteLine(properties.Length);
        Console.WriteLine(properties[0].PropertyType.Name);
        properties = birthableInterface.GetProperties();
        Console.WriteLine(properties.Length);
        Console.WriteLine(properties[0].PropertyType.Name);
        string name = Console.ReadLine();
        int age = int.Parse(Console.ReadLine());
        string id = Console.ReadLine();
        string birthdate = Console.ReadLine();
        IIdentifiable identifiable = new Citizen(name, age, id, birthdate);
        IBirthable birthable = new Citizen(name, age, id, birthdate);
    }
}
```

Задача 4.10. Телефония

Имате бизнес – производство на мобилни телефони. Но нямате софтуерни разработчици, затова се обаждате на няколко приятели и ги молите за помощ. Те вече са се съгласили и сте започнали работа. Проекта се състои от един главен модел – Смартфон. Всеки смартфон трябва да има функционалности за свързване с други телефони и достъпване на Интернет.

Тези ваши приятели са доста заети, затова решавате да напишете кода сам. Ето го вашето задължително задание:

Имате модел - Смартфон и две отделни функционалности – обаждане и достъпване на Интернет. Накрая трябва да получите един клас и два интерфейса.

Вход

Входът се задава от конзолата. Той съдържа два реда:

- Първи ред: телефонни номера (като низ), разделени с интервали.
- Втори ред: сайтове (като низ), разделени от интервали.



Изход

- Първо трябва да се обадите на всички номера според реда на въвеждането им, а след това да посетите всички сайтове в реда на въвеждането им
- Функционалността за обаждане на телефоните отпечата на конзолата съобщение в следния формат:
Calling... <номер>
- Функционалността за посещение на сайт отпечата на конзолата съобщение в следния формат:
Browsing: <сайт>!
- Ако има число в списъка с URL адреси, отпечатайте: "Invalid URL!" и продължете нататък с останалите URL адреси.
- Ако има знак различен от цифра в телефонните номера, отпечатайте: "Invalid number!" и продължете към следващия номер.

Ограничения

- Всеки URL трябва да съдържа само букви и символи (Не са позволени цифри в URL адресите)

Примери

Вход	Изход
0882134215 0882134333 08992134215 0558123 3333 1 http://mon.bg http://youtube.com http://www.g00gle.com	Calling... 0882134215 Calling... 0882134333 Calling... 08992134215 Calling... 0558123 Calling... 3333 Calling... 1 Browsing: http://mon.bg! Browsing: http://youtube.com! Invalid URL!

Решение

`namespace Telephony`

```
{
    static class Console
    {
        public static void WriteLine(object text)
        {
            System.Console.ForegroundColor = ConsoleColor.Cyan;
            System.Console.WriteLine(text);
            System.Console.ResetColor();
        }
    }

    interface ICall
    {
        void Call(string[] numbers);
    }
}
```



```
.... interface IBrowse
{
    void Browse(string[] sites);
}

class SitesAndCalls : ICall, IBrowse
{
    public void Browse(string[] sites)
    {
        for (int i = 0; i < sites.Length; i++)
            if (sites[i].Any(a => char.IsNumber(a)))
                Program.WriteLine("Invalid URL!");
            else
                Program.WriteLine("Browsing: " + sites[i]);
    }

    public void Call(string[] numbers)
    {
        for (int i = 0; i < numbers.Length; i++)
            if (ulong.TryParse(numbers[i], out ulong useless))
                Program.WriteLine("Calling... " + numbers[i]);
            else
                Program.WriteLine("Invalid number!");
    }
}

public class Program
{
    public static void Main()
    {
        var sAc = new SitesAndCalls();
        ICall call = sAc;
        call.Call(System.Console.ReadLine().Split(' '));
        IBrowse browse = sAc;
        browse.Browse(System.Console.ReadLine().Split(' '));
    }
}
```

Задача 4.11. Работници

Създайте абстрактен клас BaseEmployee, който притежава:

- Поле employeeID, което пази идентификационния номер на работника като низ.
- Поле employeeName за отбелязване на името на работника
- Поле employeeAddress за отбелязване на адреса по местоживее на работника
- Конструктор, който приема три параметъра и ги присвоява на съответните полета, изброени по-горе
- Метод Show(), който отпечата информация за работника.



- Абстрактен метод `CalculateSalary(int workingHours)`, който ще изчислява заплатата за работника, като се приема параметър – брой изработени часове
- Абстрактен метод `GetDepartment()`, който връща името на звеното от фирмата, в което работи работника

Създайте клас `FullTimeEmployee`, който наследява `BaseEmployee`, като този клас има:

- Поле `employeePosition`, което пази позицията, на която е назначен работника
- Поле `employeeDepartment`, което пази отдела, в който е назначен работника
- Конструктор с нем параметър – `employeeID`, `employeeName`, `employeeAddress`, `employeePosition`, `employeeDepartment`, който извиква конструктора на суперкласа, а след това присвоява стойностите за двете полета от този клас
- Презаписан метод `Show()`, който извиква метод `Show()` от базовия клас, а след това отпечата допълнително два реда – за позицията и отдела
- Дефиниция за абстрактния метод `CalculateSalary(int workingHours)`, като този метод връща сума, според следната формула: $250 + \text{workingHours} * 10.80$.
- Дефиниция за абстрактния метод `GetDepartment()`, като този метод връща стойността записана в `employeeDepartment`

Създайте клас `ContractEmployee`, който наследява `BaseEmployee`, като този клас има:

- Поле `employeeTask`, което пази задачата, с която този работник е назначен като контрактор
- Поле `employeeDepartment`, което пази отдела, в който е назначен работника
- Конструктор с нем параметър – `employeeID`, `employeeName`, `employeeAddress`, `employeeTask`, `employeeDepartment`, който извиква конструктора на суперкласа, а след това присвоява стойностите за двете полета от този клас
- Презаписан метод `Show()`, който извиква метод `Show()` от базовия клас, а след това отпечата допълнително ред – за задачата, с която е назначен работника
- Дефиниция за абстрактния метод `CalculateSalary(int workingHours)`, като този метод връща сума, според следната формула: $250 + \text{workingHours} * 20$.
- Дефиниция за абстрактния метод `GetDepartment()`, като този метод връща стойността записана в `employeeDepartment`

Решение

```
namespace Employees  
{
```

```
    public abstract class BaseEmployee
```



```
{
    private string id;

    public string ID
    {
        get { return id; }
        set { id = value; }
    }
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private string address;

    public string Address
    {
        get { return address; }
        set { address = value; }
    }
    public BaseEmployee(string name, string id, string address)
    {
        this.address = address;
        this.name = name;
        this.id = id;
    }
    public BaseEmployee()
    {
    }
    public void Show()
    {
        Console.WriteLine($"Employer {name} with {id} lives in {address}");
    }
    public abstract double CalculateSalary(int workingHours);
    public abstract string GetDepartment();
}

class ContractEmployee : BaseEmployee
{
    private string employeeTask;

    public string EmployeeTask
    {
        get { return employeeTask; }
        set { employeeTask = value; }
    }
    private string employeeDepartment;

    public string EmployeeDepartment
    {
        get { return employeeDepartment; }
        set { employeeDepartment = value; }
    }
}
```



```
}
    public ContractEmployee(string employeeTask, string employeeDepartment,
string name, string id, string address) : base(name, id, address)
    {
        this.employeeTask = employeeTask;
        this.employeeDepartment = employeeDepartment;
    }
    public void Show()
    {
        Console.WriteLine($"Employer {Name} with {ID} lives in {Adress} and
works in {employeeDepartment} and is currently working on {employeeTask}");
    }

    public override double CalculateSalary(int workingHours)
    {
        return 250 + workingHours * 20;
    }

    public override string GetDepartment()
    {
        return employeeDepartment;
    }
}

public class FullTimeEmployee : BaseEmployee
{
    private string employeePosition;

    public string EmployeePosition
    {
        get { return employeePosition; }
        set { employeePosition = value; }
    }

    private string employeeDepartment;

    public string EmployeeDepartment
    {
        get { return employeeDepartment; }
        set { employeeDepartment = value; }
    }

    public FullTimeEmployee(string employeeDepartment, string employeePosition,
string name, string id, string address) : base(name, id, address)
    {
        this.employeeDepartment = employeeDepartment;
        this.employeePosition = employeePosition;
    }
    public override double CalculateSalary(int workingHours)
    {
        return 250 + workingHours * 10.80;
    }

    public override string GetDepartment()
    {
```



```
        return employeeDepartment;
    }
    public void Show()
    {
        Console.WriteLine($"Employer {Name} with {ID} lives in {Adress} and
works as {employeePosition} in {employeeDepartment}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        ContractEmployee employee = new ContractEmployee("Math Problems", "Second
Floor", "Gosho", "0123", "Sliven");
        employee.Show();
        Console.WriteLine( employee.CalculateSalary(48));
    }
}
```

Задача 4.12. Животинско царство

Както всички знаем, животните обичат да са шумни. Хората пък обичат да ги дресират. Именно с това е свързана настоящата задача. Вашата цел е да създадете:

Интерфейс *IMakeNoise*

- Този интерфейс трябва да съдържа сигнатурата на метод `string MakeNoise()`

Интерфейс *IMakeTrick*

- Този интерфейс трябва да съдържа сигнатурата на метод `string MakeTrick()`

Абстрактен клас *Animal*

- Този клас трябва да имплементира `IMakeNoise` и `IMakeTrick`.
- Класът трябва да съдържа полета за `name` и `age`
- Конструктор, който приема 2 параметъра – име и възраст и задава стойностите им за полетата
- Имплементация като виртуален метод на `MakeNoise()`, която да отпечата съобщението: `My name is <name>. I am <years> old.`
- Имплементация като виртуален метод на `MakeTrick()`, която да отпечата съобщението: `Look at my trick!`

Клас *Cat*

- Този клас трябва да наследява `Animal`.
- Конструктор, който приема 2 параметъра – име и възраст и извиква базовия си конструктор
- Метод `MakeNoise()`, който отпечата съобщението: `"Meow!"`. След това извикайте `MakeNoise()` за базовия клас.



- Метод MakeTrick(), който отпечатава: No trick for you! I'm too lazy!

Клас Dog

- Този клас трябва да наследява Animal.
- Конструктор, който приема 2 параметъра – име и възраст и извиква базовия си конструктор
- Метод MakeNoise(), който отпечатава съобщението: Woof! След което извиква MakeNoise() за базовия клас
- Метод MakeTrick(), който отпечатава: Hold my paw, human!

Решение

```
namespace AnimalKingdom
{
    interface IMakeNoise
    {
        void MakeNoise();
    }

    interface IMakeTrick
    {
        void MakeTrick();
    }

    abstract class Animal : IMakeNoise, IMakeTrick
    {
        protected string name;
        protected int age;

        public Animal(string name, int age)
        {
            this.name = name;
            this.age = age;
        }

        public virtual void MakeNoise()
        {
            Console.WriteLine("My name is " + name + ". I am " + age + " years
old.");
        }

        public virtual void MakeTrick()
        {
            Console.WriteLine("Look at my trick!");
        }
    }

    class Cat : Animal
    {
        public Cat(string name, int age) : base(name, age) { }

        public virtual void MakeNoise()
        {
            Console.WriteLine("Meow!");
        }
    }
}
```



```
        base.MakeNoise();
    }

    public virtual void MakeTrick() {
        Console.WriteLine("No trick for you! I'm too lazy!");
    }
}

class Dog : Animal
{
    public Dog(string name, int age) : base(name, age) { }

    public override void MakeNoise()
    {
        Console.WriteLine("Woof!");
        base.MakeNoise();
    }

    public override void MakeTrick()
    {
        Console.WriteLine("Hold my paw, human!");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Dog doge = new Dog("vanko", 2);

        doge.MakeNoise();
        doge.MakeTrick();
    }
}
```

Задача 4.13. Граничен контрол

Бъдещето е! Вие сте владетел на тоталитарно деспотично общество населено с граждани и роботи и понеже се страхувате от безредици, решавате да имплементирате сериозен контрол върху това кой влиза в града ви. Вашите войници проверяват Id-тата на всеки, който влиза и излиза.

Ще получите неизвестно количество редове от конзолата до получаване на командата "End". На всеки ред ще има информация за гражданин или робот, който се опитва да влезе в града във формат "<name> <age> <id>" за граждани и "<model> <id>" за роботи. След командата за край, на следващия ред ще получите номер, който показва на колко завършват фалшивите Id номера, всички граждани или роботи с фалшиви Id трябва да бъдат арестувани.

Изходът от програмата трябва да съдържа всички Id-та на арестуваните, като всяко е на отделен ред.



Вход

Входът идва от конзолата. Параметрите на всяка команда ще бъдат разделени с по един интервал.

Примери

Вход	Изход
Pesho 22 9010101122 MK-13 558833251 MK-12 33283122 End 122	9010101122 33283122
Toncho 31 7801211340 Penka 29 8007181534 IV-228 999999 Stamat 54 3401018380 KKK-666 80808080 End 340	7801211340

Решение

```
namespace BorderControl
{
    public abstract class Data
    {
        private string id;

        private string name;

        public string ID { get { return id; } }

        public Data(string id, string name)
        {
            this.id = id;
            this.name = name;
        }
    }

    class Citizen : Data
    {
        private int age;

        public Citizen(string id, string name, int age) : base(id, name)
        {
            this.age = age;
        }
    }

    class Robot : Data
    {
        public Robot(string id, string name) : base(id, name)
        {
        }
    }
}
```



```
        // nope
    }
}

class Program
{
    static void Main(string[] args)
    {
        List<Robot> robots = new List<Robot>();
        List<Citizen> citizens = new List<Citizen>();
        var line = Console.ReadLine().Split().ToArray();
        while (line.Count() > 1)
        {
            if (line.Count() == 2) robots.Add(new Robot(line[1], line[0]));
            else citizens.Add(new Citizen(line[0], line[2],
int.Parse(line[1])));
            line = Console.ReadLine().Split().ToArray();
        }
        string searching = Console.ReadLine();
        Console.WriteLine(string.Join("\n", robots.Select(x =>
x.ID).ToArray().Where(x => x.Substring(x.Length - 3, 3) == searching).ToArray()));
        Console.WriteLine(string.Join("\n", citizens.Select(x =>
x.ID).ToArray().Where(x => x.Substring(x.Length - 3, 3) == searching).ToArray()));
    }
}
```

Задача 4.14. Рожден ден

Известен факт е, че хората празнуват рождениците си дни. Известно е също, че някои хора празнуват и рождениците дни дори на своите домашни любимци. Модифицирайте програмата от предната задача, така че да добавите дати на раждане към гражданите си и добавете клас Pet, домашните любимци имат име и дата на раждане. Разпределете повторещата се функционалност в интерфейси и ги имплементирайте в класовете си.

Ще получите неизвестно количество редове от конзолата до получаване на командата "End". На всеки ред ще има информация във формат "Citizen <name> <age> <id> <birthdate>" за граждани, "Robot <model> <id>" за роботи или "Pet <name> <birthdate>" за домашни любимци. След командата за край, на следващия ред ще получите номер, който е конкретна година, като вашата задача е да отпечатате всички рождени дати от тази година на всички граждани и домашни любимци във формат ден/месец/година.

Примери

Вход	Изход
Citizen Pesho 22 9010101122 10/10/1990 Pet Sharo 13/11/2005 Robot MK-13 558833251 End 1990	10/10/1990



Citizen Stamat 16 0041018380 01/01/2000 Robot MK-10 12345678 Robot PP-09 00000001 Pet Topcho 24/12/2000 Pet Kosmat 12/06/2002 End 2000	01/01/2000 24/12/2000
Robot VV-XYZ 11213141 Citizen Penka 35 7903210713 21/03/1979 Citizen Kane 40 7409073566 07/09/1974 End 1975	<empty output>

Решение

namespace Birthday

```
{  
    public abstract class Data  
    {  
        private string id;  
        private string name;  
        public string ID { get { return id; } }  
        public Data(string id, string name)  
        {  
            this.id = id;  
            this.name = name;  
        }  
    }  
  
    class Citizen : Data  
    {  
        private int age;  
        public string birthDate { get; }  
        public Citizen(string name, int age, string id, string birthDate) : base(id,  
name)  
        {  
            this.age = age;  
            this.birthDate = birthDate;  
        }  
    }  
  
    class Robot : Data  
    {  
        public Robot(string id, string name) : base(id, name)  
        {  
            //nope  
        }  
    }  
  
    interface INameAndBirthDate  
    {  
        string Name { get; }  
    }  
}
```



```
        string BirthDate { get; }
    }

    class Pet : INameAndBirthDate
    {
        private string name;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        private string birthDate;

        public string BirthDate
        {
            get { return birthDate; }
            set { birthDate = value; }
        }
        public Pet(string name, string birthDate)
        {
            this.name = name;
            this.birthDate = birthDate;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            List<Robot> robots = new List<Robot>();
            List<Citizen> citizens = new List<Citizen>();
            List<Pet> pets = new List<Pet>();
            var line = Console.ReadLine().Split().ToArray();
            while (line.Count() > 1)
            {
                switch (line[0])
                {
                    case "Citizen": citizens.Add(new Citizen(line[1],
int.Parse(line[2]), line[3], line[4]));break;
                    case "Robot": robots.Add(new Robot(line[2], line[1]));break;
                    case "Pet": pets.Add(new Pet(line[1], line[2]));break;
                }

                line = Console.ReadLine().Split().ToArray();
            }
            string searching = Console.ReadLine();
            Console.WriteLine(string.Join("\n", pets.Select(x =>
x.BirthDate).ToArray().Where(x => x.Substring(x.Length - 4, 4) ==
searching).ToArray()));
            Console.WriteLine(string.Join("\n", citizens.Select(x =>
x.birthDate).ToArray().Where(x => x.Substring(x.Length - 4, 4) ==
searching).ToArray()));
        }
    }
}
```



}

Задача 4.15. Недостиг на храна

Вашето тоталитарно деспотично общество страда от недостиг на храна, затова се появяват много бунтовници. Разширете кода от предната задача с нова функционалност, за да решите тази.

Дефинирайте клас Rebel с name, age и group (низ), имената са уникални – няма да има 2 бунтовници/граждани или бунтовник и гражданин с едно и също име. Дефинирайте интерфейс IBuyer, който дефинира метод BuyFood() и свойство Food, което да е цяло число. Имплементирайте IBuyer интерфейса в Citizen и Rebel класа, като и бунтовниците и гражданите започват с 0 храна, когато бунтовник купи храна, неговата стойност за Food се увеличава с 5, когато гражданин купи храна, неговата стойност за Food се увеличава с 10.

На първия ред от входа ще получите цяло число N – броят на хората, на всеки от следващите N реда ще получите информация в един от следните формати: "<name> <age> <id> <birthdate>" за гражданин или "<name> <age><group>" за бунтовник. След N реда ще получавате имена на хора, които са купили храна, като всяко име ще е на отделен ред. Ще получавате имена, докато не получите команда "End". Забележете, че не всички имена може да са валидни. В случай, че името е невалидно – трябва да го игнорирате.

Изход

Изходът се състои от само един ред, на който трябва да отпечатате общото количество закупена храна.

Примери

Вход	Изход
2 Pesho 25 8904041303 04/04/1989 Stanco 27 WildMonkeys Pesho Gosho Pesho End	20



4 Stamat 23 TheSwarm Toncho 44 7308185527 18/08/1973 Joro 31 Terrorists Penka 27 881222212 22/12/1988 Jiraf Joro Jiraf Joro Stamat Penka End	25
---	----

Решение

namespace FoodShortages

```
{  
    public abstract class Data  
    {  
        private string id;  
        private string name;  
        public string ID { get { return id; } }  
        public Data(string id, string name)  
        {  
            this.id = id;  
            this.name = name;  
        }  
    }  
  
    interface IBuyer  
    {  
        void BuyFood();  
        int Food { get; set; }  
    }  
  
    interface INameAndBirthDate  
    {  
        string Name { get; }  
        string BirthDate { get; }  
    }  
  
    class Citizen : Data, IBuyer  
    {  
        private int age;  
        public string birthDate { get; }  
        public int Food { get; set; }  
  
        public Citizen(string name, int age, string id, string birthDate) : base(id,  
name)  
        {  
            this.age = age;  
            this.birthDate = birthDate;  
            Food = 0;  
            Name = name;  
        }  
    }  
}
```



```
}
public string Name { get; set; }
public void BuyFood()
{
    Food += 10;
}
}

class Robot : Data, INameAndBirthDate
{
    public Robot(string id, string name, string b, int age) : base(id, name)
    {
        //nope
        this.Age = age;
        this.BirthDate = b;
    }
    public string BirthDate { get; set; }
    public int Age { get; set; }

    public string Name => throw new NotImplementedException();
}

class Rebel : IBuyer
{
    public Rebel(string name, string group, int age)
    {
        this.age = age;
        this.name = name;
        this.group = group;
        this.Food = 0;
    }
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private int age;

    public int Age
    {
        get { return age; }
        set { age = value; }
    }
    private string group;

    public string Group
    {
        get { return group; }
        set { group = value; }
    }
}

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



```
        public void BuyFood()
        {
            Food += 5;
        }
    }

    class Pet:INameAndBirthDate
    {
        private string name;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        private string birthDate;

        public string BirthDate
        {
            get { return birthDate; }
            set { birthDate = value; }
        }
        public Pet(string name,string birthDate)
        {
            this.name = name;
            this.birthDate = birthDate;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            List<Rebel> rebels = new List<Rebel>();
            List<Citizen> citizens = new List<Citizen>();
            int n = int.Parse(Console.ReadLine());
            for (int i = 0; i < n; i++)
            {
                var line = Console.ReadLine().Split().ToArray();
                if (line.Count() == 4)
                {
                    rebels.Add(new Rebel(line[0], line[2], int.Parse(line[1])));
                }
                else citizens.Add(new Citizen(line[0], int.Parse(line[1]), line[2],
line[3]));
                line = Console.ReadLine().Split().ToArray();
            }
            string s = Console.ReadLine();
            int food = 0;
            while(s!="End")
            {
                if (rebels.Select(x => x.Name).Where(x => x == s).Count() != 0) food
+= 5;
            }
        }
    }
```



```
0) food += 10;
    else if(citizens.Select(x => x.Name).Where(x => x == s).Count() !=
        s = Console.ReadLine();
    }
    Console.WriteLine(food);
}
}
```

Тема 5. Полиморфизъм

Задача 5.1. Фигури

Създайте йерархия на класа, започвайки с абстрактен клас Shape:

- Абстрактни методи:
 - calculatePerimeter(): double
 - calculateArea(): double
- Виртуални методи:
 - Draw(): string

Наследете класа Shape с двата дъщерни класа:

- Rectangle
- Circle

Всеки от тях трябва да има:

- Полета:
 - Дължина и ширина за Rectangle
 - Радиус за Circle
- Капсулация за полетата
- Публичен конструктор
- Конкретни методи за изчисления (обиколка и лице)
- Презаписани методи за рисуване

Решение

namespace Figures

```
{
    public abstract class Shape
    {
        public abstract double Perimeter();

        public abstract double Area();

        public virtual string Draw()
        {
            return "Drawing";
        }
    }

    public sealed class Circle : Shape
    {

```



```
public double Radius { get; set; }

public Circle(double radius)
{
    Radius = radius;
}

public override double Perimeter()
{
    return 2 * Math.PI * Radius;
}

public override double Area()
{
    return Math.PI * Math.Pow(Radius, 2);
}

public override string Draw()
{
    return base.Draw() + " Circle";
}
}

public sealed class Rectangle : Shape
{
    public double Width { get; set; }

    public double Height { get; set; }

    public Rectangle(double width, double height)
    {
        Width = width;
        Height = height;
    }

    public override double Perimeter()
    {
        return 2 * (Height * Width);
    }

    public override double Area()
    {
        return Height * Width;
    }

    public override string Draw()
    {
        return base.Draw() + " Rectangle";
    }
}

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

```




```
// Circle
Shape circle = new Circle(3.0);
Console.WriteLine(circle.Draw());
Console.WriteLine("Area = {0:f2}", circle.Area());
Console.WriteLine("Perimeter = {0:f2}", circle.Perimeter());

// Rectangle
Shape rectangle = new Rectangle(3.0, 4.0);
Console.WriteLine(rectangle.Draw());
Console.WriteLine("Area = {0:f2}", rectangle.Area());
Console.WriteLine("Perimeter = {0:f2}", rectangle.Perimeter());
    }
}
```

Задача 5.2. Животинско царство 2

След като построихме първото животинско царство, сега ще добавим някои новости. Този път вашата цел е да създадете:

Интерфейс *IMakeNoise*

- Този интерфейс трябва да съдържа сигнатурата на метод `string MakeNoise()`

Интерфейс *IMakeTrick*

- Този интерфейс трябва да съдържа сигнатурата на метод `string MakeTrick()`

Интерфейс *IAntimal*

- Този клас трябва да имплементира *IMakeNoise* и *IMakeTrick*.
- Интерфейсът трябва да съдържа метод `Perform()`

Клас *Cat*

- Този клас трябва да имплементира *IAntimal*.
- Метод `MakeNoise()`, който отпечата съобщението: "Meow!"
- Метод `MakeTrick()`, който отпечата: "No trick for you! I'm too lazy!"
- Метод `Perform()`, който извиква първо `MakeNoise()`, а после `MakeTrick()`

Клас *Dog*

- Този клас трябва да имплементира *IAntimal*.
- Метод `MakeNoise()`, който отпечата съобщението: "Woof!"
- Метод `MakeTrick()`, който отпечата: "Hold my paw, human!"
- Метод `Perform()`, който извиква първо `MakeNoise()`, а после `MakeTrick()`

Клас *Trainer*

- Този клас трябва да има поле *IAntimal entity*.
- Конструктор, от който да се получава обекта с животното, което ще гресираме
- Метод `Make()`, който да извиква `Perform()` метода на съответното *entity*



Решение

```
namespace AnimalKindom2
{
    public interface IMakeNoise
    {
        string MakeNoise();
    }

    public interface IMakeTrick
    {
        string MakeTrick();
    }

    public interface IAnimal : IMakeNoise, IMakeTrick
    {
        void Perform();
    }

    public class Cat : IAnimal
    {
        public string MakeNoise()
        {
            return "Meow!";
        }

        public string MakeTrick()
        {
            return "No tricks for you! I'm too lazy!";
        }

        public void Perform()
        {
            Console.WriteLine("Noise: {0}", MakeNoise());
            Console.WriteLine("Trick: {0}", MakeTrick());
        }
    }

    public class Dog : IAnimal
    {
        public string MakeNoise()
        {
            return "Woof!";
        }

        public string MakeTrick()
        {
            return "Hold my bone, human!";
        }

        public void Perform()
        {
            Console.WriteLine("Noise: {0}", MakeNoise());
            Console.WriteLine("Trick: {0}", MakeTrick());
        }
    }
}
```



```
}

public class Trainer
{
    private IAnimal animal;

    public IAnimal Animal
    {
        get { return animal; }
        set
        {
            if (value == null)
            {
                throw new ArgumentException("Animal can not be null!");
            }
            animal = value;
            Console.WriteLine("Training: {0}", animal.GetType().ToString());
        }
    }

    public Trainer(IAnimal animal)
    {
        Animal = animal;
    }

    public void Make()
    {
        Animal.Perform();
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Trainer trainer = new Trainer(new Cat());
        trainer.Make();

        trainer.Animal = new Dog();
        trainer.Make();
    }
}
```

Задача 5.3. Превозни средства

Напишете програма, която моделира 2 превозни средства (Car и Truck). Трябва да може да симулирате шофиране и презареждане на превозните средства. Car и truck имат количество гориво, консумация на гориво в литър за км и могат да бъдат управлявани на дадено разстояние и презаредени с определено количество гориво. Но през лятото и двете превозни средства използват климатик и тяхната консумация за км е завишена с 0.9 литра за Car и с 1.6 литра за Truck. Също така камионът има малка дупка в резервоара и когато се зарежда получава само 95% от



горивото. Колата няма проблеми със зареждането и получава всичкото гориво. Ако превозното средство не може да измине даденото разстояние, горивото му не се променя.

Вход

- На първи ред – информация за колата във формат {Car {fuel quantity} {liters per km}}
- На втори ред – информация за камиона във формат {Truck {fuel quantity} {liters per km}}
- На трети ред – брой команди N, които ще бъдат подадени на следващите N реда
- На следващите N реда – команди във формат:
 - Drive Car {distance}
 - Drive Truck {distance}
 - Refuel Car {liters}
 - Refuel Truck {liters}

Изход

След всяка Drive команда отпечатайте дали колата/камионът може да пропътува разстоянието, като използвате следния формат при успех:

Car/Truck travelled {distance} km

Или при неуспех:

Car/Truck needs refueling

Накрая отпечатайте оставащото гориво за колата и камиона закръглено до 2 знака след запетаята във формат:

Car: {liters}
Truck: {liters}

Примери

Вход	Изход
Car 15 0.3 Truck 100 0.9 4 Drive Car 9 Drive Car 30 Refuel Car 50 Drive Truck 10	Car travelled 9 km Car needs refueling Truck travelled 10 km Car: 54.20 Truck: 75.00
Car 30.4 0.4 Truck 99.34 0.9 5 Drive Car 500 Drive Car 13.5	Car needs refueling Car travelled 13.5 km Truck needs refueling Car: 113.05 Truck: 109.13



Refuel Truck 10.300 Drive Truck 56.2 Refuel Car 100.2	
---	--

Решение

namespace Vehicles

```
{
    public interface IVehicle
    {
        double Fuel { get; }

        double LitersPerKm { get; }

        double Distance { get; }

        void Drive(double km);

        void Refuel(double litres);
    }

    public class Car : IVehicle
    {
        public double Fuel
        {
            get { return fuel; }
            set { fuel = value; }
        }
        private double fuel;

        public double LitersPerKm
        {
            get { return litersperkm; }
            set { litersperkm = value; }
        }
        private double litersperkm;

        public double Distance
        {
            get { return distance; }
            set { distance = value; }
        }
        private double distance;

        public Car(double fuel, double litersPerKm)
        {
            Fuel = fuel;
            LitersPerKm = litersPerKm + 0.9;
            Distance = 0;
        }

        public void Drive(double km)
        {
            var travel = litersperkm * km;
            if (travel <= fuel)
```



```
        {
            Fuel -= travel;
            Console.WriteLine("Car traveled " + km + " km");
            Distance += km;
        }
        else
        {
            Console.WriteLine("Car needs refueling.");
        }
    }

    public void Refuel(double litres)
    {
        Fuel += litres; // 100%
    }
}

public class Truck : IVehicle
{
    public double Fuel
    {
        get { return fuel; }
        set { fuel = value; }
    }
    private double fuel;

    public double LitersPerKm
    {
        get { return litersperkm; }
        set { litersperkm = value; }
    }
    private double litersperkm;

    public double Distance
    {
        get { return distance; }
        set { distance = value; }
    }
    private double distance;

    public Truck(double fuel, double litersPerKm)
    {
        Fuel = fuel;
        LitersPerKm = litersPerKm + 1.6;
        Distance = 0;
    }

    public void Drive(double km)
    {
        var travel = litersperkm * km;
        if (travel <= fuel)
        {
            Fuel -= travel;
            Console.WriteLine("Truck traveled " + km + " km");
            Distance += km;
        }
    }
}
```



```
        }
        else
        {
            Console.WriteLine("Truck needs refueling.");
        }
    }

    public void Refuel(double litres)
    {
        Fuel += (litres * 0.95); // 95%
    }
}

public static class Console
{
    public static void WriteLine(string line)
    {
        var color = System.Console.ForegroundColor;
        System.Console.ForegroundColor = ConsoleColor.Green;
        System.Console.WriteLine(line);
        System.Console.ForegroundColor = color;
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        // Car [fuel quantity] [liters per km]
        var input =
System.Console.ReadLine().Split().Skip(1).Select(double.Parse).ToArray();
        Car car = new Car(input[0], input[1]);

        // Truck [fuel quantity] [liters per km]
        input =
System.Console.ReadLine().Split().Skip(1).Select(double.Parse).ToArray();
        Truck truck = new Truck(input[0], input[1]);

        int N = int.Parse(System.Console.ReadLine());
        for (int i = 0; i < N; i++)
        {
            var cmd = System.Console.ReadLine().Split().ToArray();

            if (cmd[0] == "Drive")
            {
                var distance = double.Parse(cmd[2]);
                if (cmd[1] == "Car")
                {
                    car.Drive(distance);
                }
                if (cmd[1] == "Truck")
                {
                    truck.Drive(distance);
                }
            }
        }
    }
}
```



```
        if (cmd[0] == "Refuel")
        {
            var liters = double.Parse(cmd[2]);
            if (cmd[1] == "Car")
            {
                car.Refuel(liters);
            }
            if (cmd[1] == "Truck")
            {
                truck.Refuel(liters);
            }
        }
    }

    Console.WriteLine($"Car: {car.Fuel:F2}");
    Console.WriteLine($"Truck: {truck.Fuel:F2}");
}
}
```

Задача 5.4. Превозни средства 2

Използвайте решението на предната задача като стартова точка и добавете нова функционалност. Добавете ново превозно средство – Bus. Сега всяко превозно средство има капацитет на резервоара и количество на горивото, което не може да падне под 0 (ако количеството гориво падне под 0, отпечатайте на конзолата "Fuel must be a positive number").

Колата и автобуса не могат да се заредят с гориво повече от техния капацитет на резервоара. Ако се опитате да сложите повече гориво в резервоара от наличното място, отпечатайте "Cannot fit fuel in tank" и не добавяйте гориво в резервоара.

Добавете нова команда за автобуса. Автобусът може да пътува със или без хора. Ако автобусът пътува с хора, то климатикът трябва да е включен и неговата консумация на гориво за километър се увеличава с 1.4 литра. Ако в автобуса няма хора, то климатикът ще е изключен и консумацията не се увеличава.

Вход

- На първите три реда въвеждате информация за превозните средства във формат:
Vehicle {initial fuel quantity} {liters per km} {tank capacity}
- На четвъртия ред – броят на командите N, които ще бъдат подадени на следващите N реда
- На следващите N реда – команди във формата
 - Drive Car {distance}
 - Drive Truck {distance}
 - Drive Bus {distance}
 - DriveEmpty Bus {distance}



- Refuel Car {liters}
- Refuel Truck {liters}
- Refuel Bus {liters}

Изход

След всяка Drive команда отпечатайте дали колата/камионът/автобусът може да пропътува това разстояние във формат при успех:

Car/Truck/Bus travelled {distance} km

Или при неуспех:

Car/Truck/Bus needs refueling

Ако даденото гориво е ≤ 0 , отпечатайте "Fuel must be a positive number".

Ако даденото гориво, не може да се вмести в резервоара, отпечатайте "Cannot fit in tank"

Накрая, отпечатайте оставащото гориво за колата, камиона и автобуса, закръглени до 2 знака след запетаята във формат:

Car: {liters} Truck: {liters} Bus: {liters}

Пример

Вход	Изход
Car 30 0.04 70	Cannot fit fuel in tank
Truck 100 0.5 300	Bus travelled 10 km
Bus 40 0.3 150	Cannot fit fuel in tank
8	Bus needs refueling
Refuel Car -10	Car: 30.00
Refuel Truck 0	Truck: 1050.00
Refuel Car 10	Bus: 23.00
Refuel Car 300	
Drive Bus 10	
Refuel Bus 1000	
DriveEmpty Bus 100	
Refuel Truck 1000	

Решение

```
namespace Vehicles2
{
    public interface IVehicle
    {
        double Fuel { get; }

        double FuelCapacity { get; }

        double LitersPerKm { get; }
    }
}
```



```
double Distance { get; }

void Drive(double km);

void Refuel(double litres);
}

public class Bus : IVehicle
{
    public double Fuel
    {
        get { return fuel; }
        set { fuel = value; }
    }
    private double fuel;

    public double LitersPerKm
    {
        get { return litersperkm; }
        set { litersperkm = value; }
    }
    private double litersperkm;

    public double FuelCapacity
    {
        get { return fuelCapacity; }
        set { fuelCapacity = value; }
    }
    private double fuelCapacity;

    public double Distance
    {
        get { return distance; }
        set { distance = value; }
    }
    private double distance;

    public bool Passangers { get; set; }

    public Bus(double fuel, double liters, double capacity)
    {
        Fuel = fuel;
        LitersPerKm = liters;
        FuelCapacity = capacity;
        Distance = 0;
        Passangers = false;
    }

    public void Drive(double km)
    {
        var travel = litersperkm * km;
        if (travel <= fuel)
        {
            Fuel -= travel;
            Console.WriteLine("Bus traveled " + km + " km");
        }
    }
}
```



```
        Distance += km;
    }
    else
    {
        Console.WriteLine("Bus needs refueling");
    }
}

public void Refuel(double litres)
{
    if (litres < 0 || litres > fuelCapacity)
    {
        Console.WriteLine("Cannot fit fuel in tank");
    }
    else Fuel += litres;
}

}

public class Car : IVehicle
{
    public double Fuel
    {
        get { return fuel; }
        set { fuel = value; }
    }
    private double fuel;

    public double LitersPerKm
    {
        get { return litersperkm; }
        set { litersperkm = value; }
    }
    private double litersperkm;

    public double FuelCapacity
    {
        get { return fuelCapacity; }
        set { fuelCapacity = value; }
    }
    private double fuelCapacity;

    public double Distance
    {
        get { return distance; }
        set { distance = value; }
    }
    private double distance;

    public Car(double fuel, double liters, double capacity)
    {
        Fuel = fuel;
        LitersPerKm = liters + 0.9;
        FuelCapacity = capacity;
        Distance = 0;
    }
}
```



```
public void Drive(double km)
{
    var travel = litersperkm * km;
    if (travel <= fuel)
    {
        Fuel -= travel;
        Console.WriteLine("Car traveled " + km + " km");
        Distance += km;
    }
    else
    {
        Console.WriteLine("Car needs refueling.");
    }
}

public void Refuel(double litres)
{
    if (litres < 0 || litres > fuelCapacity)
    {
        Console.WriteLine("Cannot fit fuel in tank");
    }
    else Fuel += litres; // 100%
}

public class Truck : IVehicle
{
    public double Fuel
    {
        get { return fuel; }
        set { fuel = value; }
    }
    private double fuel;

    public double LitersPerKm
    {
        get { return litersperkm; }
        set { litersperkm = value; }
    }
    private double litersperkm;

    public double FuelCapacity
    {
        get { return fuelCapacity; }
        set { fuelCapacity = value; }
    }
    private double fuelCapacity;

    public double Distance
    {
        get { return distance; }
        set { distance = value; }
    }
    private double distance;
}
```



```
public Truck(double fuel, double liters, double capacity)
{
    Fuel = fuel;
    LitersPerKm = liters + 1.6;
    FuelCapacity = capacity;
    Distance = 0;
}

public void Drive(double km)
{
    var travel = litersperkm * km;
    if (travel <= fuel)
    {
        Fuel -= travel;
        Console.WriteLine("Truck traveled " + km + " km");
        Distance += km;
    }
    else
    {
        Console.WriteLine("Truck needs refueling.");
    }
}

public void Refuel(double litres)
{
    if (litres < 0 || litres > fuelCapacity)
    {
        Console.WriteLine("Cannot fit fuel in tank");
    }
    else Fuel += (litres * 0.95); // 95%
}

public static class Console
{
    public static void WriteLine(string line)
    {
        var color = System.Console.ForegroundColor;
        System.Console.ForegroundColor = ConsoleColor.Green;
        System.Console.WriteLine(line);
        System.Console.ForegroundColor = color;
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        // Car [initial fuel quantity] [liters per km] [tank capacity]
        var input =
System.Console.ReadLine().Split().Skip(1).Select(double.Parse).ToArray();
        Car car = new Car(input[0], input[1], input[2]);

        // Truck [initial fuel quantity] [liters per km] [tank capacity]
```



```
input =
System.Console.ReadLine().Split().Skip(1).Select(double.Parse).ToArray();
Truck truck = new Truck(input[0], input[1], input[2]);

// Bus [initial fuel quantity] [liters per km] [tank capacity]
input =
System.Console.ReadLine().Split().Skip(1).Select(double.Parse).ToArray();
Bus bus = new Bus(input[0], input[1], input[2]);

// N команди
int N = int.Parse(System.Console.ReadLine());
for (int i = 0; i < N; i++)
{
    var cmd = System.Console.ReadLine().Split().ToArray();

    if (cmd[0] == "Drive")
    {
        var distance = double.Parse(cmd[2]);
        if (cmd[1] == "Car")
        {
            car.Drive(distance);
        }
        if (cmd[1] == "Truck")
        {
            truck.Drive(distance);
        }
        if (cmd[1] == "Bus")
        {
            bus.Drive(distance);
        }
    }

    else if (cmd[0] == "Refuel")
    {
        var liters = double.Parse(cmd[2]);
        if (cmd[1] == "Car")
        {
            car.Refuel(liters);
        }
        if (cmd[1] == "Truck")
        {
            truck.Refuel(liters);
        }
        if (cmd[1] == "Bus")
        {
            bus.Refuel(liters);
        }
    }

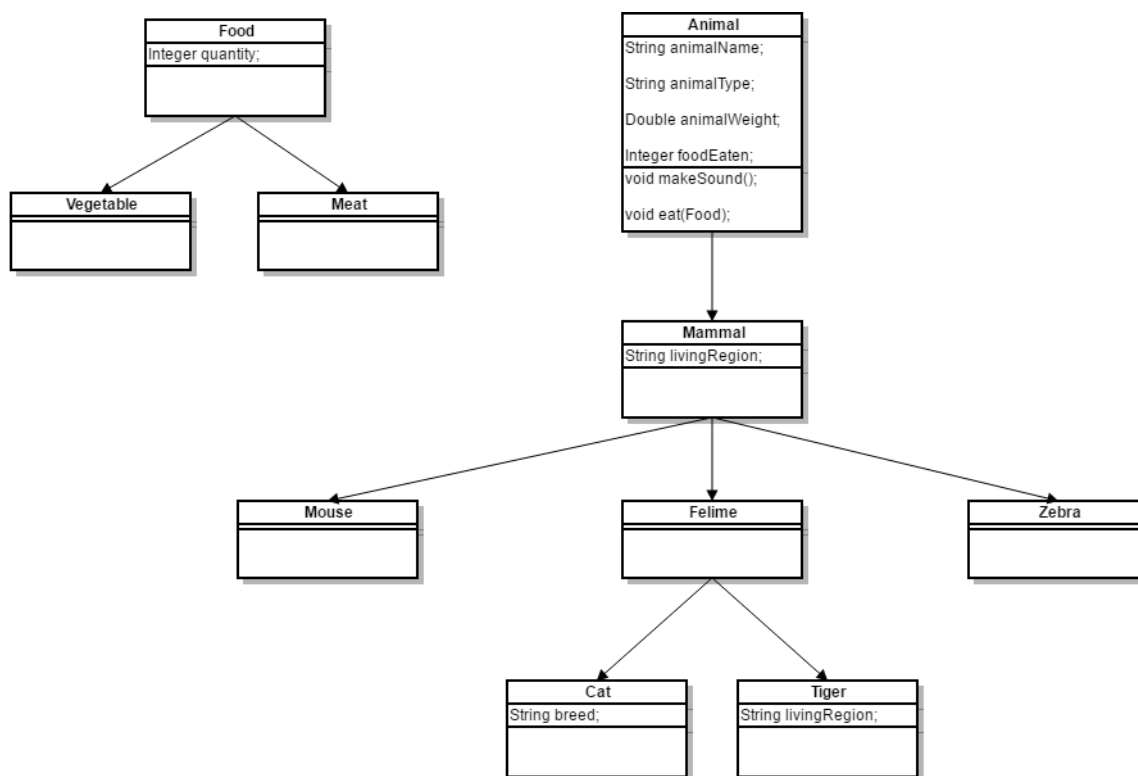
    // DriveEmpty Bus [distance]
    else
    {
        bus.Passangers = false;
        bus.Drive(double.Parse(cmd[2]));
    }
}
```



```
    }  
  
    Console.WriteLine($"Car: {car.Fuel:F2}");  
    Console.WriteLine($"Truck: {truck.Fuel:f2}");  
    Console.WriteLine($"Bus: {bus.Fuel:f2}");  
    }  
}  
}
```

Задача 5.5. Ферма за животни

Вашата задача е да създадете йерархия от класове като тази на диаграмата по-долу. Всички класове, освен Vegetable, Meat, Mouse, Tiger, Cat и Zebra, трябва да са абстрактни. Презаписвайте метод `ToString()`.



Входът трябва да се прочете от конзолата. Всеки четен ред ще съдържа информация за животно в следния формат:

```
{AnimalType} {AnimalName} {AnimalWeight} {AnimalLivingRegion} [{CatBreed} =  
Only if its cat]
```

На нечетните редове ще получите информация за храната, която трябва да дадете на животното. Редът ще съдържа `FoodType` и количество разделено от интервал.



Трябва да направите логиката, която определя дали животното ще яде предоставената му храна. Мишката и зебрата трябва да проверят дали храната им е Зеленчук. Ако е – те ще я ядат. В противен случай, трябва да отпечатайте съобщение в следния формат:

```
{AnimalType} are not eating that type of food!
```

Котките ядат каква да е храна, но тигрите приемат само месо. Ако се даде зеленчук на тигъра, отпечатайте съобщение като това отгоре на конзолата.

Презапишете ToString метода, така че да отпечата информация за животното във формата:

```
{AnimalType} [{AnimalName}, {CatBreed}, {AnimalWeight}, {AnimalLivingRegion},  
{FoodEaten}]
```

След като въведете информация за животно и храна, извикайте MakeSound метода за текущото животно и след това го нахранете. В края отпечатайте целия обект и продължете да четете информация за следващото животно/храна. Входът ще продължи докато не получите команда "End".

Вход	Изход
Cat Gray 1.1 Home Persian Vegetable 4 End	Meowwww Cat[Gray, Persian, 1.1, Home, 4]
Tiger Typcho 167.7 Asia Vegetable 1 End	ROAAR!!! Tigers are not eating that type of food! Tiger[Typcho, 167.7, Asia, 0]
Zebra Doncho 500 Africa Vegetable 150 End	Zs Zebra[Doncho, 500, Africa, 150]
Mouse Jerry 0.5 Anywhere Vegetable 0 End	SQUEEEAAK! Mouse[Jerry, 0.5, Anywhere, 0]

Решение

```
namespace AnimalFarm  
{  
    public abstract class Animal  
    {  
        public string animalName { get; }  
        public string animalType { get; }  
    }  
}
```




```
public double animalWeight { get; }
public int foodEaten { get; }

public virtual void MakeSound()
{
    // nope
}

public void Eat()
{
    // nope
}

public Animal(string animalType, string name, double weight)
{
    this.animalType = animalType;
    this.animalName = name;
    this.animalWeight = weight;
}

}

public class Cat : Felime
{
    public string breed { get; set; }

    public Cat(string animalType, string name, double weight, string
livingRegion, string breed) : base(animalType, name, weight, livingRegion)
    {
        this.breed = breed;
    }

    public override void MakeSound()
    {
        Console.WriteLine("Meowwww");
    }

    public override string ToString()
    {
        return
${animalType}[{animalName},{breed},{animalWeight},{livingRegion}];
    }
}

public static class Console
{
    public static void WriteLine(string line)
    {
        var color = System.Console.ForegroundColor;
        System.Console.ForegroundColor = ConsoleColor.Green;
        System.Console.WriteLine(line);
        System.Console.ForegroundColor = color;
    }
}

public abstract class Felime : Mammal
```



```
{
    public Felime(string animalType, string name, double weight, string
livingRegion) : base(animalType, name, weight, livingRegion)
    {
        // nope
    }
}

public abstract class Food
{
    public int quantity { get; }

    public Food(int quantity)
    {
        this.quantity = quantity;
    }
}

public abstract class Mammal : Animal
{
    public string livingRegion { get; }

    public Mammal(string animalType, string name, double weight, string
livingRegion) : base(animalType, name, weight)
    {
        this.livingRegion = livingRegion;
    }
}

public class Meat : Food
{
    public Meat(int quantity) : base(quantity)
    {
        // nope
    }
}

public class Mouse : Mammal
{
    public Mouse(string animalType, string name, double weight, string
livingRegion) : base(animalType, name, weight, livingRegion)
    {
        // nope
    }

    public override void MakeSound()
    {
        Console.WriteLine("SQUEEEAAK!");
    }

    public override string ToString()
    {
        return $"{animalType}[{animalName},{animalWeight},{livingRegion}";
    }
}
```



```
public class Tiger : Felime
{
    public Tiger(string animalType, string name, double weight, string
livingRegion) : base(animalType, name, weight, livingRegion)
    {
        // nope
    }

    public override void MakeSound()
    {
        Console.WriteLine("ROAAR!!!");
    }

    public override string ToString()
    {
        return $"{animalType}[{animalName},{animalWeight},{livingRegion}];"
    }
}

public class Vegetable : Food
{
    public Vegetable(int quantity) : base(quantity)
    {
        // nope
    }
}

public class Zebra : Mammal
{
    public Zebra(string animalType, string name, double weight, string
livingRegion) : base(animalType, name, weight, livingRegion)
    {
        // nope
    }

    public override void MakeSound()
    {
        Console.WriteLine("Zs");
    }

    public override string ToString()
    {
        return $"{animalType}[{animalName},{animalWeight},{livingRegion}];"
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        var line = System.Console.ReadLine().Split(' ').ToArray(); ;
        Animal animal = new Tiger("", "", 0, ""); Food food = new Meat(0);
        for (int i = 1; line[0] != "End"; i++)
        {
            if (i % 2 != 0)
```



```
        switch (line[0])
        {
            case "Cat": animal = new Cat(line[0], line[1],
double.Parse(line[2]), line[3], line[4]); break;
            case "Mouse": animal = new Mouse(line[0], line[1],
double.Parse(line[2]), line[3]); break;
            case "Tiger": animal = new Tiger(line[0], line[1],
double.Parse(line[2]), line[3]); break;
            case "Zebra": animal = new Zebra(line[0], line[1],
double.Parse(line[2]), line[3]); break;
        }
    else
    {
        switch (line[0])
        {
            case "Vegetable": food = new Vegetable(int.Parse(line[1]));
break;
            case "Meat": food = new Meat(int.Parse(line[1])); break;
        }
        switch (animal.animalType)
        {
            case "Cat":
                animal.MakeSound();
                Console.WriteLine(animal.ToString() + "," +
food.quantity);
                break;
            case "Mouse":
                animal.MakeSound();
                if (line[0] == "Meat")
                    Console.WriteLine($"{animal.animalType}s are not
eating that type of food!");
                Console.WriteLine(animal.ToString() + "," +
food.quantity);
                break;
            case "Tiger":
                animal.MakeSound();
                if (line[0] == "Vegetable")
                    Console.WriteLine($"{animal.animalType}s are not
eating that type of food!");
                Console.WriteLine(animal.ToString() + "," +
food.quantity);
                break;
            case "Zebra":
                animal.MakeSound();
                if (line[0] == "Meat")
                    Console.WriteLine($"{animal.animalType}s are not
eating that type of food!");
                Console.WriteLine(animal.ToString() + "," +
food.quantity);
                break;
        }
    }
    line = System.Console.ReadLine().Split(' ').ToArray();
}
```



}

Тема 6. Работа с обекти

Задача 6.1. Библиотека

Създайте клас Book, който трябва да има три публични свойства:

- string Title
- int Year
- List<string> Authors

Авторите може да са анонимни, един или повече. Book трябва да има само един конструктор.

Създайте клас Library, който да съхранява колекция книги и да имплементира интерфейса IEnumerable<Book> и да съдържа в себе си списък с книги:

- List<Book> books

Library може да е инициализиран без книги или с какъвто и да е брой книги и трябва да има само един конструктор.

Решение

```
namespace Library
{
    public class Book
    {
        public string Title { get; set; }

        public int Year { get; set; }

        public IReadOnlyList<string> Authors { get; private set; }

        public Book(string title, int year, params string[] authors)
        {
            Title = title;
            Year = year;
            Authors = authors;
        }

        public override string ToString()
        {
            return string.Format("{0}. {1}. {2}.", Title, Year, string.Join(", ",
Authors));
        }
    }

    public class Library
    {
        public List<Book> books { get; set; }

        public Library(params Book[] books)
        {

```



```
        this.books = new List<Book>(books);
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Book bookOne = new Book("Animal Farm", 2003, "George Orwell");
        Book bookTwo = new Book("The Documents in the Case", 2002, "Dorothy
Sayers", "Robert Eustace");
        Book bookThree = new Book("The Documents in the Case", 1930);

        Library libraryOne = new Library();
        Library libraryTwo = new Library(bookOne, bookTwo, bookThree);
    }
}
```

Задача 6.2. Итератор за библиотека

Разширете решението на предишната задача. В класа Library създайте вложен клас LibraryIterator, който трябва да имплементира интерфейса IEnumerable<Book>. Опишете да имплементирате тялото на двата наследени метода сами. Ще ви трябват още два члена:

- List<Book> books
- int currentIndex

Сега ще можете да обхождате Library в Main() метода.

Решение

```
namespace LibraryIterator
{
    public class Book
    {
        public string Title { get; set; }

        public int Year { get; set; }

        public IReadOnlyList<string> Authors { get; private set; }

        public Book(string title, int year, params string[] authors)
        {
            Title = title;
            Year = year;
            Authors = authors;
        }

        public override string ToString()
        {
            return string.Format("{0}. {1}. {2}.", Title, Year, string.Join(", ",
Authors));
        }
    }
}
```



```
public class Library : IEnumerable<Book>
{
    public List<Book> books { get; set; }

    public Library(params Book[] books)
    {
        this.books = new List<Book>(books);
    }

    public IEnumerator<Book> GetEnumerator()
    {
        foreach (Book book in books)
        {
            yield return book;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}

public class LibraryIterator : IEnumerator<Book>
{
    private readonly List<Book> books;

    private int CurrentBook;

    public LibraryIterator(IEnumerable<Book> books)
    {
        this.Reset();
        this.books = new List<Book>(books);
    }

    public Book Current => books[CurrentBook];

    object IEnumerator.Current => this.Current;

    public bool MoveNext() => ++CurrentBook < books.Count();

    public void Reset() => CurrentBook = -1;

    public void Dispose()
    {
        // nope
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Library library = new Library
        (
```



```
        new Book("Animal Farm", 2003, "George Orwell"),  
        new Book("The Documents in the Case", 2002, "Dorothy Sayers",  
"Robert Eustace"),  
        new Book("The Documents in the Case", 1930)  
    );  
  
    foreach (var book in library)  
    {  
        Console.WriteLine(book);  
    }  
}  
}
```

Задача 6.3. ListyIterator

Създайте шаблонен клас ListyIterator, той трябва да получава чрез своя конструктор колекцията, която ще обхожда. Трябва да съхранява елементите в списък. Класът трябва да има три основни функции:

- Move – да мести позицията на вътрешния индекс към следващия индекс в списъка; методът трябва да връща true ако успешно я е преместил и false ако няма следващ индекс.
- HasNext – да връща стойност true ако има следващ индекс и стойност false ако индексът е вече на последния елемент от списъка.
- Print – трябва да отпечата елемента на дадения вътрешен индекс; изпълнението на Print върху колекция без елементи трябва да хвърля подходящо изключение със съобщението "Invalid Operation!".

По подразбиране вътрешният индекс трябва да сочи към нулевия индекс в списъка. Програмата ви трябва да поддържа следните команди:

Команда	Връщан тип	Описание
Create {e1 e2 ...}	void	Създава ListyIterator от указаната колекция. В случай на команда Create без подадени елементи да създава ListyIterator с празна колекция.
Move	boolean	Тази команда трябва да мести вътрешния индекс към следваща позиция.
Print	void	Тази команда трябва да отпечата елемента на дадения вътрешен индекс.
HasNext	boolean	Връща дали колекцията има следващ елемент.



END	void	Спира входа.
-----	------	--------------

Вход

Входът ще идва от конзолата като няколко реда с команди. Първият ред винаги ще бъде единствената команда Create във входа. Последната получена команда винаги ще бъде единствената команда END.

Изход

За всяка команда от входа (с изключение на командите END и Create) отпечатайте нейният резултат на конзолата, всяка команда на нов ред. В случай на командата Move или HasNext да се отпечата върнатата стойност на методите; в случай на командата Print не е необходимо да правите нищо допълнително, тъй като самият метод трябва да се отпечата на конзолата. Програмата трябва да прихваща всички изключения, хвърлени заради валидация (например извикването на Print при празна колекция), и да отпечата съобщенията им.

Ограничения

- Винаги ще има само 1 Create команда и тя винаги ще е първата подадена.
- Номерата на получените команди ще са между [1...100].
- Последната команда винаги ще е единствената команда END.

Примери

Вход	Изход
Create Print END	Invalid Operation!
Create Stefcho Goshky HasNext Print Move Print END	True Stefcho True Goshky
Create 1 2 3 HasNext Move HasNext HasNext Move HasNext END	True True True True True False

Решение

`namespace ListyIterator`



```
{
    public class ListyIterator<T> : IEnumerable<T>
    {
        private int index = 0;
        private List<T> list;

        public ListyIterator(params T[] listy)
        {
            this.Create(listy);
        }

        public void Create(params T[] listy)
        {
            this.list = new List<T>(listy);
        }

        public bool Move()
        {
            if (!HasNext()) return false;
            else
            {
                this.index = this.index + 1;
                return true;
            }
        }

        public string Print()
        {
            if (this.list.Count == 0)
            {
                throw new Exception("Invalid Operation");
            }
            return this.list[this.index].ToString();
        }

        public bool HasNext()
        {
            if (this.index + 1 > this.list.Count - 1) return false;
            else return true;
        }

        public IEnumerator<T> GetEnumerator()
        {
            for (int i = 0; i < this.Count(); i++)
            {
                yield return this.list[i];
            }
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return this.GetEnumerator();
        }

        public void PrintAll()
    }
}
```



```
{
    foreach (var item in list)
    {
        Console.Write(item + " ");
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        ListyIterator<string> list = new ListyIterator<string>();
        var line = Console.ReadLine().Split().ToArray();
        while (line[0] != "END")
        {
            switch (line[0])
            {
                case "PrintAll":
                    list.PrintAll();
                    break;
                case "Create":
                    list.Create(line.Skip(1).ToArray());
                    break;
                case "HasNext":
                    Console.WriteLine(list.HasNext());
                    break;
                case "Move":
                    Console.WriteLine(list.Move());
                    break;
                case "Print":
                    Console.WriteLine(list.Print());
                    break;
            }
            line = Console.ReadLine().Split().ToArray();
        }
    }
}
```

Задача 6.4. Колекция

Като използваме ListyIterator-а от предишната задача, разширете го чрез имплементиране на интерфейса IEnumerable<T>; имплементирайте ръчно всички методи, изисквани от интерфейса (използвайте yield return за метода GetEnumerator()). Добавете нова команда PrintAll, която трябва да обхожда с foreach колекцията и да отпечата всички елементи на един ред и отделени с интервал.

Вход

Входът ще идва от конзолата като няколко реда с команди. Първият ред винаги ще бъде единствената команда Create във входа. Последната получена команда винаги ще бъде единствената команда END.



Изход

За всяка команда от входа (с изключение на командите END и Create) отпечатайте резултата от тази команда на конзолата, всяка команда на нов ред. В случай на командата Move или HasNext да се отпечата върнатата стойност на методите; в случай на командата Print не е необходимо да правите нищо допълнително, тъй като самият метод трябва да се отпечата на конзолата. В случай на командата PrintAll трябва да отпечатате всички елементи на един ред, отделени един от друг с пауза. Програмата трябва да улавя всички изключения хвърлени заради валидиране и да отпечата съобщенията им.

Ограничения

НЕ използвайте метода GetEnumerator() от базовия клас. Използвайте своя собствена имплементация, която да ползва "yield return"

Винаги ще има само 1 Create команда и тя винаги ще е първата подадена.

Номерата на получените команди ще са между [1...100].

Последната команда винаги ще е единствената команда END.

Примери

Вход	Изход
Create 1 2 3 4 5 Move PrintAll END	True 1 2 3 4 5
Create Stefcho Goshky Peshu PrintAll Move Move Print HasNext END	Stefcho Goshky Peshu True True Peshu False

Решение

Задача 6.5. Стек

След като вече сте преминали курса с основни алгоритми, сега имате задачата да създадете своя версия на стек. Познавате структурата на стековете. Има колекция, която да съхранява елементите, и две функции (но не от функционалното програмиране) - за добавяне на елемент (push) и за извличане (pop). Имайте в предвид, че първият елемент който се извлича е последният в колекцията. Push метода добавя елемент в края на колекцията, а pop метода връща най-горния елемент и го премахва.



Напишете своя лична имплементация на `Stack<T>` и реализирайте интерфейса `IEnumerable<T>`. Вашата имплементация на метода `GetEnumerator()` трябва да спазва правилата на абстрактния тип данни `Stack` (т.е. да връща елементите в обратния ред на този, по който са добавени в стека).

Вход

Входът ще идва от конзолата като няколко реда с команди. Командите ще са само `push` и `pop`, следвани от цели числа за командата `push` и никакъв друг вход за командата `pop`.

Формат:

- `Push {element1}, {element2}, ... {elementN}` – добавя дадените елементи към стека
- `Pop` – премахва последния вмъкнат елемент от стека

Изход

Когато получите `END`, с въвеждането се спира. Обходете два пъти стека с `foreach` и отпечатайте всички елементи, всеки на нов ред.

Ограничения

Елементите в командата `push` ще са валидни цели числа между `[2-32...232-1]`.

Командите винаги ще са валидни (винаги ще са `Push`, `Pop` или `END`).

Ако някоя команда `Pop` не може да се изпълни както се очаква (например ако няма елементи в стека), отпечатайте на конзолата: `"No elements"`.

Примери

Вход	Изход
Push 1, 2, 3, 4 Pop Pop END	2 1 2 1
Push 1, 2, 3, 4 Pop Push 1 END	1 3 2 1 1 3 2 1
Push 1, 2, 3, 4 Pop Pop	No elements



Pop Pop Pop END	
--------------------------	--

Решение

```
namespace Stack
{
    public class MyEnumerator<T> : IEnumerator<T>
    {
        private int position;
        private Stack<T> stack;

        public MyEnumerator(Stack<T> stack)
        {
            this.stack = stack;
            position = -1;
        }
        public void Dispose()
        {
        }

        public void Reset()
        {
            position = -1;
        }

        public bool MoveNext()
        {
            position++;
            return position <= stack.count;
        }

        Object IEnumerator.Current
        {
            get
            {
                return stack.array[position];
            }
        }
        public T Current
        {
            get
            {
                return stack.array[position];
            }
        }
    }

    public class Stack<T> : IEnumerable<T>
    {
        public T[] array;
        public int count;
    }
}
```



```
public Stack()
{
    array = new T[5];
    count = -1;
}

public void Push(T item)
{
    count++;
    if (count > array.Length)
    {
        Array.Resize<T>(ref array, count + 1);
    }

    array[count] = item;
}

public T Pop()
{
    if (count > 0)
    {
        count--;
        return array[count + 1];
    }
    else
    {
        return array[count];
    }
}

public IEnumerator<T> GetEnumerator()
{
    return new MyEnumerator<T>(this);
}

IEnumerator IEnumerable.GetEnumerator()
{
    return new MyEnumerator<T>(this);
}

}

internal class Program
{
    static void Main(string[] args)
    {
        Stack<string> stack = new Stack<string>();

        List<string> input = Console.ReadLine().Split(' ').ToList();

        while (input[0] != "END")
        {
            switch (input[0])
            {
                case "Push":
                    input.RemoveAt(0);
```



```
        for (int i = 0; i < input.Count; i++)
        {
            input[i] = input[i].Replace(",", "");
            stack.Push(input[i]);
        }

        break;
    case "Pop":
        stack.Pop();
        break;
    }

    input = Console.ReadLine().Split(' ').ToList();
}

List<string> reversed = new List<string>();

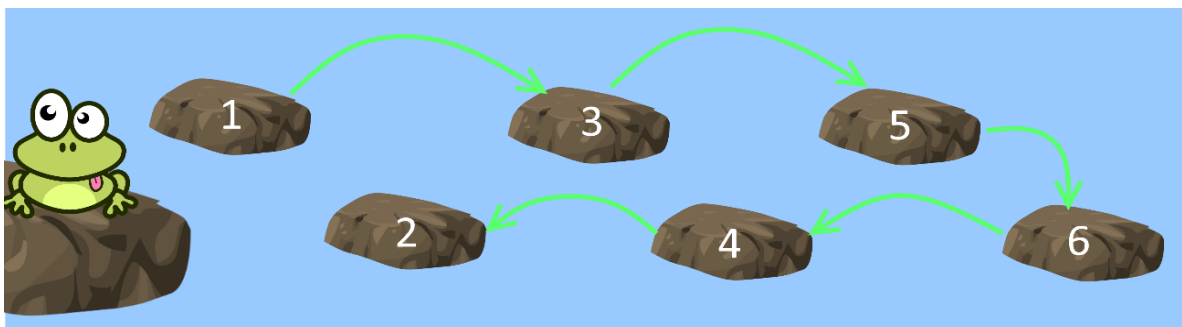
foreach (string item in stack)
{
    reversed.Add(item);
}

reversed.Reverse();

foreach (string item in reversed)
{
    Console.WriteLine(item);
}
foreach (string item in reversed)
{
    Console.WriteLine(item);
}
}
}
```

Задача 6.6. Жабче

Да играем на една игра. Имате една мъничка Жаба и Езеро с пътечка от камъни. Всеки камък има номер. Нашата жаба трябва да пресече езерото по пътеката и после да се върне. Но има някои правила за скачане по камъните. Първо, жабата трябва да скочи на всички четни позиции върху камъните във възходящ ред, а после на всички нечетни позиции, но в низходящ ред. Редът на камъните и техните номера ще бъдат зададени на първия ред на входа. След това трябва да отпечатате реда на камъните, по който нашата жаба е скочила от единия на другия.



Опитайте да постигнете тази функционалност чрез създаване на клас Lake (той ще съдържа номерата на всички камъни подред), който имплементира интерфейса `IEnumerable<int>` и предефинира неговите `GetEnumerator()` методи.

Примери

Вход	Изход
1, 2, 3, 4, 5, 6, 7, 8	1, 3, 5, 7, 8, 6, 4, 2
1, 2, 3, 4, 5	1, 3, 5, 4, 2
13, 23, 1, -8, 4, 9	13, 1, 4, 9, -8, 23

Решение

```
namespace Frog
```

```
{  
    public class Lake : IEnumerable<int>  
    {  
        private int[] rocks;  
  
        public Lake(int[] rocks)  
        {  
            this.rocks = rocks;  
        }  
  
        public IEnumerator<int> GetEnumerator()  
        {  
            for (int i = 0; i < rocks.Length; i += 2)  
            {  
                yield return rocks[i];  
            }  
  
            for (int i = rocks.Length - (rocks.Length % 2 == 0 ? 1 : 2); i >= 0; i -  
= 2)  
            {  
                yield return rocks[i];  
            }  
        }  
    }  
}
```



```
IEnumerator IEnumerable.GetEnumerator()
{
    return this.GetEnumerator();
}

internal class Program
{
    static void Main(string[] args)
    {
        var a = Console.ReadLine()
            .Split(new char[] { ' ', ',' },
StringSplitOptions.RemoveEmptyEntries)
            .Select(int.Parse).ToArray();

        Lake lake = new Lake(a);

        Console.WriteLine(string.Join(", ", lake));
    }
}
```

Задача 6.7. *Клиника за домашни любимци

Вие сте млад и амбициозен собственик на Pet Clinics Holding. Искате от подчинените си да създадат програма, която да съхранява цялата информация за животните в базата данни. Всеки любимец трябва да има име, възраст и вид.

Вашето приложение трябва да поддържа няколко основни операции като например създаване на любимец, създаване на клиника, добавяне на любимец в клиника, изписване на любимец от клиника, отпечатване на информация за конкретна стая в клиниката и отпечатване на информация за всички стаи в клиниката.

Клиниките трябва да имат нечетен брой стаи; опитът за създаване на клиника с четен брой стаи трябва да се провали и да хвърли подходящо изключение.

Рег на настаняване

Да вземем например клиника с 5 стаи. Първата стая, в която ще се лекува любимец, е централната (стая 3). Така че редът, по който се приемат животни, е: първият любимец отива в централната (3) стая, после следващите влизат първо в лявата (2) и после в дясната (4) стая. Последните стаи, в които влизат животни, са стаи 1 и 5. В случай, че стая вече е заета я прескачаме и отиваме в следващата в горепосочения ред. Вашата задача е да моделирате приложението и да се погрижите да поддържа някои команди.

- Първият любимец влиза в стая 3. -> 1 2 3 4 5
- След това вторият влиза в стая 2. -> 1 2 3 4 5



- Третият любимец ще влезе в стая 4. -> 1 2 3 4 5
- А последните два ще отидат в стаи 1 и 5. -> 1 2 3 4 5

След като сме покрили добавянето на любимци, време е да намерим начин да ги изписваме. Процесът на изписване не е толкова прост; когато методът за изписване е повикан, започваме от централната стая (3) и продължаваме надясно (4, 5... и така нататък) докато не открием любимец или не достигнем последната стая. Ако достигнем последната стая, започваме от първата (1) и отново се движим надясно докато не достигнем централната стая (3). Ако е открит любимец го премахваме от колекцията, спираме по-нататъшно търсене и връщаме true; ако НЕ е открит любимец, операцията връща false.

Когато е повикана команда print за стая, ако стаята съдържа любимец го отпечатайте на един ред във формата "<име на любимеца> <възраст на любимеца> <вид на любимеца>". Ако пък стаята е празна, отпечатайте "Room empty". Когато е повикана команда print за клиника, трябва да отпечатаме всички стаи в клиниката по ред на техните номера.

Команди

Команда	Връщане	Описание
Create Pet {име} {възраст} {вид}	boolean	Създава любимец с указаните име и възраст. (true ако операцията е успешна и false ако не е)
Create Clinic {име} {стаи}	void	Създава клиника с указаните име и брой стаи (ако стаите не са нечетен брой, хвърля изключение)
Add {име на любимец} {име на клиника}	boolean	Тази команда трябва да добави дадения любимец в указаната клиника (true ако операцията е успешна и false ако не е).
Release {име на клиника}	boolean	Тази команда трябва да изпише любимец от указаната клиника (true ако операцията е успешна и false ако не е).
HasEmptyRooms {име на клиника}	boolean	Връща дали клиниката има празни стаи. (true ако има и false ако няма).



Print {име на клиника}	void	Тази команда трябва да отпечата всички стаи в указаната клиника, подредени по номера.
Print {име на клиника} {стая}	void	Отпечатва на един ред съдържанието на конкретната стая.

Вход

На първия ред ще ви бъде дадено цяло число N – броят команди, които ще получите. На всеки от следващите N на брой редове ще получите команда. Командите и параметрите винаги ще са верни (командите Add, Release, HasEmptyRooms и Print винаги ще са за съществуващи клиники/любимци) с изключение на броя стаи в командата Create Clinic, който може да е всяко валидно цяло число между 1 и 101.

Изход

За всяка команда с булев връщан тип данни, получена през входа, трябва да отпечатате върнатата от нея стойност на нов ред. В случай, че метод хвърли изключение (например при опит за създаване на клиника с четен брой стаи или за добавяне на любимец, който не съществува), трябва да го прихванете и вместо това да отпечатате "Invalid Operation!". Командата Print с клиника и стая трябва да отпечатва информация за тази стая в уточнения по-горе формат. Командата Print само с клиника трябва да отпечатва информация за всяка стая в клиниката по ред на номерата.

Ограничения

- Броят команди N ще е валидно цяло число между [1...1000]; не е нужно да го проверявате изрично.
- Имената на любимци, имената на клиники и вида ще са низове, съдържащи само азбучни символи, с дължина между [1...50] символа.
- Възрастта на любимците ще е положително цяло число между [1...100].
- Клиничните стаи ще са положително цяло число между [1...101].
- Номерът на стая в команда Print винаги ще е между 1 и броя стаи в тази клиника.
- Входът ще се състои само от верни команди и те винаги ще имат правилен брой параметри.

Пример

Вход	Изход
9	Invalid Operation!
Create Pet Gosho 7 Cat	True
Create Clinic Rezovo 4	False
Create Clinic Rizovo 1	True
HasEmptyRooms Rizovo	False
Release Rizovo	False



Add Gosho Rizovo HasEmptyRooms Rizovo Create Pet Sharo 2 Dog Add Sharo Rizovo	
8 Create Pet Gosho 7 Cat Create Pet Sosho 1 Cata Create Clinic Rezovo 5 Add Gosho Rezovo Add Sosho Rezovo Print Rezovo 3 Release Rezovo Print Rezovo	True True Gosho 7 Cat True Room empty Sosho 1 Cata Room empty Room empty Room empty

Решение

```
namespace PetsClinic
{
    public class Clinic
    {
        private int busyRooms;

        public int BusyRooms
        {
            get { return busyRooms; }
            set { busyRooms = 0; }
        }

        private string name;

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

        private int numberOfRooms;

        public int NumberOfRooms
        {
            get { return numberOfRooms; }
            set
            {
                if (value % 2 == 0)
                {
                    Console.WriteLine("Invalid operation!");
                }
                else
                {
                    numberOfRooms = value;
                }
            }
        }
    }
}
```



```
}

public Room[] rooms;

List<Pet> pets;

public Clinic()
{
    rooms = new Room[numberOfRooms];
    pets = new List<Pet>();
}

public Clinic(string name, int numberOfRooms)
{
    this.name = name;
    NumberOfRooms = numberOfRooms;
    rooms = new Room[numberOfRooms + 1];
    for (int i = 1; i <= numberOfRooms; i++)
    {
        rooms[i] = new Room(i, true);
    }
    pets = new List<Pet>();
}

public bool AddPet(Pet pet)
{
    int i = numberOfRooms / 2 + 1;
    int counter = 0;
    while (counter < numberOfRooms)
    {
        if (busyRooms < numberOfRooms)
        {
            if (counter % 2 == 0)
            {
                if (rooms[i - counter].IsEmpty)
                {
                    rooms[i - counter] = new Room(i - counter, false, pet);
                    pets.Add(pet);
                    busyRooms++;
                    return true;
                }
            }
            else if (rooms[i + counter].IsEmpty)
            {
                rooms[i + counter] = new Room(i + counter, false, pet);
                pets.Add(pet);
                busyRooms++;
                return true;
            }
        }
        counter++;
    }
    else return false;
}
return true;
}
```



```
public bool ReleasePet()
{
    int i = numberOfRooms / 2 + 1;
    int counter = 0;
    while (counter < numberOfRooms)
    {
        if (busyRooms != 0)
        {
            if (counter % 2 == 0)
            {
                if (rooms[i - counter].IsEmpty == false)
                {
                    rooms[i - counter] = new Room(i - counter, true);
                    pets.Remove(rooms[i - counter].Pet);
                    busyRooms--;
                    return true;
                }
            }
            else if (rooms[i + counter].IsEmpty == false)
            {
                rooms[i + counter] = new Room(i + counter, true);
                pets.Remove(rooms[i + counter].Pet);
                busyRooms--;
                return true;
            }
        }
        counter++;
    }
    return false;
}

public class PetClinicsHolding
{
    List<Clinic> clinics = new List<Clinic>();

    List<Pet> pets = new List<Pet>();

    public void AddClinic(Clinic clinic)
    {
        clinics.Add(clinic);
    }

    public void AddPet(Pet pet)
    {
        pets.Add(pet);
    }

    public bool AddPetInClinic(string petName, string clinicName)
    {

```



```
        if (pets.Select(x => x.Name).Contains(petName) && clinics.Select(x =>
x.Name).Contains(clinicName))
        {
            Clinic currentClinic = clinics.Where(x => x.Name ==
clinicName).First();
            return currentClinic.AddPet(pets.Where(x => x.Name ==
petName).First());
        }
        else return false;
    }
    public bool RemovePetFromClinic(string clinicName)
    {
        if (clinics.Select(x => x.Name).Contains(clinicName))
        {
            Clinic currentClinic = clinics.Where(x => x.Name ==
clinicName).First();
            return currentClinic.ReleasePet();
        }
        else return false;
    }

    public bool HasEmptyRooms(string clinicName)
    {
        Clinic currentClinic = clinics.Where(x => x.Name == clinicName).First();
        if (currentClinic.BusyRooms < currentClinic.NumberOfRooms)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public void PrintClinic(string clinicName)
    {
        Clinic currentClinic = clinics.Where(x => x.Name == clinicName).First();
        for (int i = 1; i <= currentClinic.NumberOfRooms; i++)
        {
            if (currentClinic.rooms[i].IsEmpty)
            {
                Console.WriteLine("Room Empty");
            }
            else
            {
                Console.WriteLine(currentClinic.rooms[i].Pet.Name + " " +
currentClinic.rooms[i].Pet.Age + " " + currentClinic.rooms[i].Pet.Type);
            }
        }
    }

    public void PrintRoom(string clinicName, int roomNumber)
    {
        Clinic currentClinic = clinics.Where(x => x.Name == clinicName).First();
        Room room = currentClinic.rooms[roomNumber];
```




```
        Console.WriteLine(room.Pet.Name + " " + room.Pet.Age + " " +
room.Pet.Type);
    }
}

public class Pet
{
    private string name;

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

    private int age;

    public int Age
    {
        get { return age; }
        set { age = value; }
    }

    private string type;

    public string Type
    {
        get { return type; }
        set { type = value; }
    }

    public Pet()
    {
        // nope
    }

    public Pet(string name, int age, string type)
    {
        this.name = name;
        this.age = age;
        this.type = type;
    }

    public bool Create(Pet pet)
    {
        if (pet.name.Length <= 50 && pet.age <= 100 && pet.type.Length <= 50)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```



```
public class Room
{
    private int number;

    public int Number
    {
        get { return number; }
        set { number = value; }
    }

    private bool isEmpty;

    public bool IsEmpty
    {
        get { return isEmpty; }
        set { isEmpty = value; }
    }

    private Pet pet;

    public Pet Pet
    {
        get { return pet; }
        set { pet = value; }
    }

    public Room(int number, bool isEmpty = true)
    {
        this.number = number;
        this.isEmpty = isEmpty;
    }

    public Room(int number, bool isEmpty, Pet pet) : this(number, isEmpty)
    {
        this.pet = pet;
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());

        Pet pet;
        Clinic clinic;
        PetClinicsHolding clinics = new PetClinicsHolding();

        for (int i = 0; i < n; i++)
        {
            var line = Console.ReadLine().Split(' ').ToArray();
            switch (line[0])
            {
                case "Create":
```



```
        {
            switch (line[1])
            {
                case "Pet":
                {
                    pet = new Pet(line[2], int.Parse(line[3]),
line[4]);

                    Console.WriteLine(pet.Create(pet));
                    clinics.AddPet(pet);
                    break;
                }
                case "Clinic":
                {
                    clinic = new Clinic(line[2],
int.Parse(line[3]));

                    clinics.AddClinic(clinic);
                    break;
                }
            }
            break;
        }
        case "Add":
        {
            Console.WriteLine(clinics.AddPetInClinic(line[1],
line[2]));

            break;
        }
        case "Release":
        {
            Console.WriteLine(clinics.RemovePetFromClinic(line[1]));
            break;
        }
        case "HasEmptyRooms":
        {
            Console.WriteLine(clinics.HasEmptyRooms(line[1]));
            break;
        }
        case "Print":
        {
            if (line.Length == 3)
            {
                clinics.PrintRoom(line[1], int.Parse(line[2]));
            }
            else
            {
                clinics.PrintClinic(line[1]);
            }
            break;
        }
    }
}
}
```



Задача 6.8. ***Обхождане на свързан списък

Трябва да напишете своя собствена опростена имплементация на стандартен Свързан Списък (Linked List), който имплементира `IEnumerable<T>`. Списъкът трябва да поддържа операциите `Add` и `Remove`; трябва да разкрива броя елементи, които има, със свойството `Count`. Методът `Add` трябва да добавя новия елемент в края на колекцията. Методът `Remove` трябва да премахва първото появяване на елемента от началото на колекцията; ако елементът е успешно премахнат методът да връща `true`, ако пък елементът не е в колекцията методът трябва да върне `false`. Свойството `Count` трябва да върне броя елементи, които списъкът съдържа в момента.

Вход

В първия ред от входа ще получите число `N`. На всеки от следващите `N` редове ще получите команда в един от следните формати:

`Add <number>` - добавя число към вашия свързан списък.

`Remove <number>` - премахва първото появяване на числото от свързания списък. Ако няма такъв елемент, тази команда оставя колекцията непроменена.

Изход

Изходът трябва да съдържа точно 2 реда. На първия трябва да отпечатате резултата от извикването на свойството `Count` на свързания списък. На втория трябва да отпечатате всички елементи от колекцията на един ред отделени с интервали, като извикате метода `foreach` за колекцията.

Ограничения

- Всички числа във входа ще са валидни цели числа между `[2-32...232-1]`.
- Всички команди получени през входа ще бъдат валидни (ще са само `Add` или `Remove`).
- Броят хора `N` ще е положително цяло число между `[1...500]`.

Примери

Вход	Изход
5 Add 7 Add -50 Remove 3 Remove 7 Add 20	2 -50 20
6 Add 13 Add 4	4 13 20 4 4



Add 20 Add 4 Remove 4 Add 4	
--------------------------------------	--

Съвет

Може да използвате упражненията за линейни структури от данни, което ще ви помогне да имплементирате своя свързан списък. Ресурсите там би трябвало да са достатъчно ръководство как да го направите.

Решение

```
namespace LinkedList
{
    public class LinkedList<T> : IEnumerable<T>
    {
        private Node<T> head;
        private Node<T> tail;

        public int Count { get; private set; }

        public LinkedList()
        {
            head = null;
            tail = null;
            Count = 0;
        }

        public void Add(T item)
        {
            if (head == null)
            {
                Node<T> next = new Node<T>(item);
                head = next;
                tail = next;
            }
            else
            {
                Node<T> next = new Node<T>(item, tail);
                tail = next;
            }
            Count++;
        }

        public bool Remove(T item)
        {
            int index = 0;
            Node<T> current = head;

            if (EqualityComparer<T>.Default.Equals(head.Element, item))
            {
                head = head.Next;
                Count--;
                return true;
            }
        }
    }
}
```



```
    }

    while (current != null)
    {
        if (current.Element.Equals(item))
        {
            Remove(index);
            Count--;
            return true;
        }
        index++;
        current = current.Next;
    }
    return false; // Not Found
}

public object Remove(int index)
{
    object item = null;
    Node<T> current = head;
    Node<T> previous = current;
    int i = 0;
    while (current != null)
    {
        if (index == i)
        {
            item = current.Element;
            previous.Next = current.Next;
            break;
        }
        i++;
        previous = current;
        current = current.Next;
    }
    return item;
}

public IEnumerator<T> GetEnumerator()
{
    Node<T> node = head;
    while (node != null)
    {
        yield return node.Element;
        node = node.Next;
    }
}

IEnumerator IEnumerable.GetEnumerator()
{
    return GetEnumerator();
}

}

public class Node<T>
{
```



```
private T element;

public T Element
{
    get { return this.element; }
    set { this.element = value; }
}

private Node<T> next;

public Node<T> Next
{
    get { return this.next; }
    set { this.next = value; }
}

public Node(T element, Node<T> prevNode)
{
    this.element = element;
    prevNode.next = this;
}

public Node(T element)
{
    this.element = element;
    next = null;
}
}

internal class Program
{
    static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        LinkedList<string> list = new LinkedList<string>();

        for (int i = 0; i < n; i++)
        {
            string[] input = Console.ReadLine().Split().ToArray();

            switch (input[0])
            {
                case "Add":
                {
                    list.Add(input[1]);
                    break;
                }
                case "Remove":
                {
                    list.Remove(input[1]);
                    break;
                }
            }
        }
    }
}
```



```
        Console.WriteLine(list.Count);  
        foreach (var item in list)  
        {  
            Console.Write(item + " ");  
        }  
    }  
}
```

Задача 6.9. Сравнима книга

Разширете решението на предишната задача. Имплементирайте интерфейса `IComparable<Book>` в съществуващия клас `Book`. Сравнението на две книги трябва да се случва по следния ред:

- Първо ги сортирайте по възходящ хронологичен ред (по година)
- Ако две книги са публикувани в една и съща година, сортирайте ги по азбучен ред

Предефинирайте метода `ToString()` в своя клас `Book`, за да връща низ във формата:

- {заглавие} - {година}

Променете своя клас `Library` така, че да съхранява книгите в правилния ред.

Не е нужно да променяте нищо в своя `Main` метод от предишната задача освен начина на отпечатване на книга в конзолата.

Примери

Изход
The Documents in the Case - 1930 The Documents in the Case - 2002 Animal Farm - 2003

Решение

```
namespace ComparableBook  
{  
    public class Book : IComparable<Book>  
    {  
        public string Title { get; set; }  
  
        public int Year { get; set; }  
  
        public IReadOnlyList<string> Authors { get; private set; }  
  
        public Book(string title, int year, params string[] authors)  
        {  
            Title = title;  
            Year = year;  
            Authors = authors;  
        }  
    }  
}
```




```
public override string ToString()
{
    return string.Format($"{Title} - {Year}.");
}

public int CompareTo(Book other)
{
    int result = this.Year.CompareTo(other.Year);
    if (result == 0)
    {
        result = this.Title.CompareTo(other.Title);
    }
    return result;
}

public class Library : IEnumerable<Book>
{
    public List<Book> books { get; set; }

    public Library(params Book[] books)
    {
        this.books = new List<Book>(books);
        this.books = this.books
            .OrderBy(book => book.Year)
            .ThenBy(book => book.Title).ToList();
    }

    public IEnumerator<Book> GetEnumerator()
    {
        foreach (Book book in books)
        {
            yield return book;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}

public class LibraryIterator : IEnumerator<Book>
{
    private readonly List<Book> books;

    private int CurrentBook;

    public LibraryIterator(IEnumerable<Book> books)
    {
        this.Reset();
        this.books = new List<Book>(books);
    }
}
```



```
public Book Current => books[CurrentBook];

object IEnumerator.Current => this.Current;

public bool MoveNext() => ++CurrentBook < books.Count();

public void Reset() => CurrentBook = -1;

public void Dispose()
{
    // nope
}

static void Main(string[] args)
{
    Library library = new Library
    (
        new Book("Animal Farm", 2003, "George Orwell"),
        new Book("The Documents in the Case", 2002, "Dorothy Sayers",
"Robert Eustace"),
        new Book("The Documents in the Case", 1930)
    );

    foreach (var book in library)
    {
        Console.WriteLine(book);
    }
}
```

Задача 6.10. Сравняващият книги

Разширете решението на предишната задача. Създайте клас BookComparator, който да имплементира интерфейса IComparer<Book> и така да включва следния метод:

- int Compare(Book, Book)

BookComparator трябва да сравнява две книги по:

1. Заглавие – азбучен ред
2. Година на издаване на книгата – от най-нови към най-стари

Модифицирайте своя клас Library отново, така че да имплементирате новото сортиране.

Примери

Startup.cs



```
public static void Main()
{
    Book bookOne = new Book("Animal Farm", 2003, "George Orwell");
    Book bookTwo = new Book("The Documents in the Case", 2002, "Dorothy
Sayers", "Robert Eustace");
    Book bookThree = new Book("The Documents in the Case", 1930);
}
```

Изход

```
Animal Farm - 2003
The Documents in the Case - 2002
The Documents in the Case - 1930
```

Решение

```
namespace BookComparator
{
    public class Book
    {
        public string Title { get; set; }

        public int Year { get; set; }

        public IReadOnlyList<string> Authors { get; private set; }

        public Book(string title, int year, params string[] authors)
        {
            Title = title;
            Year = year;
            Authors = authors;
        }

        public override string ToString()
        {
            return string.Format("{0} - {1}.", Title, Year);
        }
    }

    public class BookComparator : IComparer<Book>
    {
        public int Compare(Book x, Book y)
        {
            int result = x.Title.CompareTo(y.Title);
            if (result == 0)
            {
                result = y.Year.CompareTo(x.Year);
            }
            return result;
        }
    }
}
```



```
public class Library : IEnumerable<Book>
{
    public List<Book> books { get; set; }

    public Library(params Book[] books)
    {
        this.books = new List<Book>(books);
        this.books = this.books
            .OrderBy(book => book.Title)
            .ThenByDescending(book => book.Year).ToList();
    }

    public IEnumerator<Book> GetEnumerator()
    {
        foreach (Book book in books)
        {
            yield return book;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}

public class LibraryIterator : IEnumerator<Book>
{
    private readonly List<Book> books;

    private int CurrentBook;

    public LibraryIterator(IEnumerable<Book> books)
    {
        this.Reset();
        this.books = new List<Book>(books);
    }

    public Book Current => books[CurrentBook];

    object IEnumerator.Current => this.Current;

    public bool MoveNext() => ++CurrentBook < books.Count();

    public void Reset() => CurrentBook = -1;

    public void Dispose()
    {
        // nope
    }
}

static void Main(string[] args)
{
    Library library = new Library
```



```
(  
    new Book("Animal Farm", 2003, "George Orwell"),  
    new Book("The Documents in the Case", 2002, "Dorothy Sayers",  
"Robert Eustace"),  
    new Book("The Documents in the Case", 1930)  
);  
foreach (var book in library)  
{  
    Console.WriteLine(book);  
}  
}
```

Задача 6.11. Сравняване на обекти

Има нещо такова като интерфейс Comparable, предполагам, че вероятно вече го знаете. Вашата задача е проста. Създайте клас Person. Всеки човек трябва да има име, възраст и град. Трябва да имплементирате интерфейса IComparable<T> и метода CompareTo. Когато сравнявате двама души, първо сравнете имената им, след това – възрастите им, а накрая – градовете им.

Вход

На всеки ред ще получавате човек във формат:

{име} {възраст} {град}

Колекционирайте ги, докато не получите "END"

След това ще получите цяло число N - Nтия човек в колекцията ви. Започвайки от 1.

Изход

На единствения ред от изхода изведете статистики: колко хора са еднакви с него, колко не са и общият брой хора във вашата колекция.

Формат: {брой еднакви хора} {брой нееднакви хора} {общ брой хора}

Ограничения

Входните имена, възрасти и адреси ще са валидни. Входното число винаги ще е валидно цяло число в интервала [2...100]

Ако няма еднакви хора, отпечатайте: "No matches"

Примери

Вход	Изход
Pesho 22 Vraca Gogo 14 Sofeto END 2	No matches
Pesho 22 Vraca Gogo 22 Vraca	2 1 3



Gogo 22 Vraca

END

2

Решение

namespace ComparablePerson

{

public class Person : IComparable<Person>

{

private string name;

public string Name

{

get { return name; }

private set { name = value; }

}

private int age;

public int Age

{

get { return age; }

private set { age = value; }

}

private string city;

public string City

{

get { return city; }

private set { city = value; }

}

public Person(string name, int age, string city)

{

this.name = name;

this.age = age;

this.city = city;

}

public int CompareTo(Person other)

{

int compared = this.Name.CompareTo(other.Name);

if (compared == 0)

{

compared = this.Age.CompareTo(other.Age);

}

if (compared == 0)

{

compared = this.City.CompareTo(other.City);

}

return compared;

}

}



```
internal class Program
{
    static void Main(string[] args)
    {
        List<Person> people = new List<Person>();
        while (true)
        {
            var cmd = Console.ReadLine().Split().ToArray();
            if (cmd[0] == "END") break;
            people.Add(new Person(cmd[0], int.Parse(cmd[1]), cmd[2]));
        }

        int n = int.Parse(Console.ReadLine());
        Person searched = people[n - 1];
        int matches = 0;

        foreach (var person in people)
        {
            if (person.CompareTo(searched) == 0)
            {
                matches++;
            }
        }
        if (matches != 1)
        {
            Console.WriteLine($"{matches} {people.Count - matches}
{people.Count}");
        }
        else
        {
            Console.WriteLine("No matches");
        }
    }
}
```

Задача 6.12. Шаблон Strategy

Интересен шаблон, за който може да сте чували, е Strategy; ако има няколко начина да се изпълни задача (например да се сортира колекция), той позволява на клиента да избере начина, който най-много подхожда на нуждите му. Известна имплементация на шаблона в C# са методите [List<T>.Sort\(\)](#) и [Array.Sort\(\)](#), които използват IComparer като аргумент.

Създайте клас Person, който съдържа име и възраст. Създайте два компаратора за Person (класове, които имплементират интерфейса IComparer<Person>). Първият компаратор трябва да сравнява хора по дължината на името им като първи параметър; ако двамата души имат имена с една и съща дължина, вместо това сравнява първата буква от имената, без да прави разлика между малки и главни букви. Вторият компаратор трябва да ги сравнява по възраст.



Създайте 2 обекта от `mun SortedSets` с елементи от `mun Person`; първият трябва да имплементира компараторът за имена, а втория да имплементира компаратора за възраст.

Вход

На първия ред от входа ще получите число `N`. На всеки от следващите `N` реда ще получите информация за хора във формата "`<name> <age>`". Добавете хората от входа и в двете сортирани колекции (те трябва да съдържат всички хора, подадени като входни данни).

Изход

Обходете с `foreach` колекциите и отпечатайте всеки човек от тях на нов ред в същия формат, в който сте го получили. Започнете с този, който имплементира компаратора за имена.

Ограничения

- Името на човек ще е низ, който съдържа само букви и цифри и ще е с дължина `[1...50]` символа.
- Възрастта на човек ще е положително цяло число между `[1...100]`.
- Броят хора `N` ще е положително цяло число между `[0...100]`.

Примери

Вход	Изход
3 Pesho 20 Joro 100 Pencho 1	Joro 100 Pesho 20 Pencho 1 Pencho 1 Pesho 20 Joro 100
5 Ivan 17 asen 33 Stoqn 25 Nasko 99 Joro 3	asen 33 Ivan 17 Joro 3 Nasko 99 Stoqn 25 Joro 3 Ivan 17 Stoqn 25 asen 33 Nasko 99

Решение

```
namespace PersonStrategy
{
    public class Person
    {
        public Person(string name, int age)
        {
            this.Name = name;
        }
    }
}
```




```
        this.Age = age;
    }

    public string Name { get; private set; }

    public int Age { get; private set; }

    public override string ToString()
    {
        return $"{this.Name} {this.Age}";
    }
}

public class PersonAgeComparator : IComparer<Person>
{
    public int Compare(Person x, Person y)
    {
        return x.Age - y.Age;
    }
}

public class PersonNameComparator : IComparer<Person>
{
    public int Compare(Person x, Person y)
    {
        var comparison = x.Name.Length - y.Name.Length;

        return comparison == 0
            ? (int)(char.ToLower(x.Name[0]) - char.ToLower(y.Name[0]))
            : comparison;
    }
}

internal class Program
{
    public static void Main()
    {
        var peopleByName = new SortedSet<Person>(new PersonNameComparator());
        var peopleByAge = new SortedSet<Person>(new PersonAgeComparator());
        FillSetsWithPeople(peopleByName, peopleByAge);
        PrintBothSets(peopleByName, peopleByAge);
    }

    private static void PrintBothSets(SortedSet<Person> peopleByName,
SortedSet<Person> peopleByAge)
    {
        var sb = new StringBuilder();

        ExtractPeopleData(sb, peopleByName);
        ExtractPeopleData(sb, peopleByAge);

        Console.Write(sb);
    }
}
```



```
private static void ExtractPeopleData(StringBuilder sb, SortedSet<Person>
peopleCollection)
{
    foreach (var person in peopleCollection)
    {
        sb.AppendLine(person.ToString());
    }
}

private static void FillSetsWithPeople(SortedSet<Person> peopleByName,
SortedSet<Person> peopleByAge)
{
    var numberOfPeople = int.Parse(Console.ReadLine());

    while (numberOfPeople > 0)
    {
        var personData = Console.ReadLine().Split();
        var name = personData[0];
        var age = int.Parse(personData[1]);

        peopleByName.Add(new Person(name, age));
        peopleByAge.Add(new Person(name, age));

        numberOfPeople--;
    }
}
```

Задача 6.13. *Логика за еднаквост

Създайте клас Person, съдържащ име и възраст. Хора с еднакви име и възраст възприемайте за един и същи; предефинирайте всички необходими методи, за да наложите тази логика. Вашият клас трябва да работи както със стандартни, така и с хеширани колекции. Създайте SortedSet и HashSet от типа Person.

Вход

На първия ред от входа ще получите число N. На всеки от следващите N реда ще получите информация за хора във формата "<name> <age>". Добавете хората от входа и в двете колекции (те трябва да съдържат всички хора, получени като информация от входа).

Изход

Изходът трябва да се състои от точно два реда. На първия трябва да отпечтатите размера на дървовидната колекция, а на втория – размера на хешираната.

Ограничения

- Името на човек ще е низ, който съдържа само букви и цифри и ще е с дължина [1...50] символа.
- Възрастта на човек ще е положително цяло число между [1...100].



- Броят хора N ще е положително цяло число между [0...100].

Примери

Вход	Изход
4 Pesho 20 Peshp 20 Joro 15 Pesho 21	4 4
7 Ivan 17 ivan 17 Stoqn 25 Ivan 18 Ivan 17 Stopn 25 Stoqn 25	5 5

Съвет

Трябва да предефинирате и метода Equals, и метода GetHashCode. Можеме да потърсим имплементация на GetHashCode онлайн – не е нужно да е съвършена, но трябва да е достатъчно добра да произведе същия хеш код за обекти с еднакви име и възраст, както и достатъчно различни хеш кодове за обекти с различни име и/или възраст.

Решение

```
namespace PersonEquality
{
    public class Person : IComparable
    {
        private string name;

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

        private int age;

        public int Age
        {
            get { return age; }
            set { age = value; }
        }

        public Person(string name, int age)
        {
            this.name = name;
            this.age = age;
        }
    }
}
```



```
public override bool Equals(object obj)
{
    Person person = (Person)obj;
    if (this.Age == person.age && this.name == person.name)
    {
        return true;
    }
    return false;
}

public override int GetHashCode()
{
    return this.name.GetHashCode() + age;
}

public int CompareTo(object obj)
{
    Person person = (Person)obj;
    if (this.Age == person.age && this.name == person.name)
    {
        return 0;
    }
    return 1;
}
}

internal class Program
{
    static void Main(string[] args)
    {
        SortedSet<Person> peopleSet = new SortedSet<Person>();
        HashSet<Person> peopleHash = new HashSet<Person>();

        int n = int.Parse(Console.ReadLine());

        for (int i = 0; i < n; i++)
        {
            var cmd = Console.ReadLine().Split().ToArray();
            Person person = new Person(cmd[0], int.Parse(cmd[1]));

            if (peopleSet.Where(x => x.Equals(person)).Count() == 0)
            {
                peopleSet.Add(person);
            }

            if (peopleHash.Where(x => x.GetHashCode() ==
person.GetHashCode()).Count() == 0)
            {
                peopleHash.Add(person);
            }
        }

        Console.WriteLine(peopleSet.Count);
        Console.WriteLine(peopleHash.Count);
    }
}
```



```
}  
}  
}
```

Задача 6.14. Крадец

Добавете към проекта си класа Hacker от кутията по-долу.

Hacker.cs

```
public class Hacker  
{  
    public string username = "securityGod82";  
    private string password = "mySuperSecretPassw0rd";  
  
    public string Password  
    {  
        get => this.password;  
        set => this.password = value;  
    }  
  
    private int Id { get; set; }  
  
    public double BankAccountBalance { get; private set; }  
  
    public void DownloadAllBankAccountsInTheWorld()  
    {  
    }  
}
```

Има един много гаден хакер, но не е много умен. Опитва се да открадне голяма сума пари и да я прехвърли в собствената си сметка. Полицията го преследва, но им трябва професионалист... Правилно – това сте вие!

Разполагате с информацията, че този хакер пази част от информацията си в `private` полета. Създайте нов клас, наречен `Spy`, и добавете в него метод, наречен **StealFieldInfo**, който да получава:

- низ – име на класа, който да разследва
- масив от низове – имена на полетата, които да разследва

След като намерите полетата, отпечатайте на конзолата:

- "Class under investigation: {имеНаКласа}"

На следващите редове отпечатайте информация за всяко поле в настоящия формат:

- "{имеНаПоле} = {стойностНаПоле}"



Използвайте StringBuilder да свържете отговора. Не променяйте нищо в "Hacker" класа!

В Main метод трябва да можете да проверите програмата си с това нарче ког:

```
public static void Main()
{
    Spy spy = new Spy();
    string result = spy.StealFieldInfo("Hacker", "username", "password");
    Console.WriteLine(result);
}
```

Пример

Изход

```
Class under investigation: Hacker
username = securityGod82
password = mySuperSecretPassw0rd
```

Решение

namespace Hacker

```
{
    public class Hacker
    {
        public string username = "securityGod82";

        public string Password
        {
            get => this.password;
            set => this.password = value;
        }
        private string password = "mySuperSecretPassw0rd";

        private int Id { get; set; }

        public double BankAccountBalance { get; private set; }

        public void DownloadAllBankAccountsInTheWorld()
        {
            // empty
        }
    }

    public class Spy
    {
        public string StealFieldInfo(string className, params string[] fieldsNames)
        {
            Console.WriteLine("Class Under Investigation: {0}", className);
        }
    }
}
```



```
Type type = Type.GetType(className);
string typeFullName = type.FullName;

FieldInfo[] fields = type.GetFields
(
    BindingFlags.Static |
    BindingFlags.Instance |
    BindingFlags.NonPublic |
    BindingFlags.Public
);

StringBuilder sb = new StringBuilder();
foreach (var field in fields)
{
    Type fieldType = type.ReflectedType;
    var testInstance = Activator.CreateInstance(type);
    var fieldValue = field.GetValue(testInstance);
    sb.Append(String.Format("{0}: {1}\n", field.Name,
fieldValue.ToString()));
}

return sb.ToString();
}

internal class Program
{
    static void Main(string[] args)
    {
        Spy spy = new Spy();
        string result = spy.StealFieldInfo("Hacker", "username", "password");
        Console.WriteLine(result);
    }
}
```

Задача 6.15. Висококачествени грешки

Вече сте експерт в Качествен Код, така че знаете какъв вид модификатори на достъп трябва да бъде зададен на членовете на класа. Трябва да сте забелязали, че нашият хакер не е запознат с тези концепции.

Създайте метод в своя Spy клас, наречен **AnalyzeAccessModifiers**(string className). Проверете всички модификатори на достъп на полетата и методите. Отпечатайте на конзолата всички грешки във формат:

Полета

- {имеНаПоле} must be private!

Getters

- {имеНаМетод} have to be public!

Setters

- {имеНаМетод} have to be private!



Използвайте StringBuilder да свържете отговора. Не променяйте нищо в "Hacker" класа!

В Main метода си би трябвало да можете да проверите програмата си със следния блок от код:

```
public static void Main()
{
    Spy spy = new Spy();
    string result = spy.AnalyzeAccessModifiers("Hacker");
    Console.WriteLine(result);
}
```

Пример

Изход
username must be private! get_Id have to be public! set_Password have to be private!

Решение

```
namespace HackerQualityCode
{
    public class Hacker
    {
        public string username = "securityGod82";
        private string password = "mySuperSecretPassw0rd";

        public string Password
        {
            get => this.password;
            set => this.password = value;
        }

        private int Id { get; set; }

        public double BankAccountBalance { get; private set; }

        public void DownloadAllBankAccountsInTheWorld()
        {
            // nope
        }

        public string AnalyzeAccessModifiers(string className)
        {
            Type myType = Type.GetType(className);
            FieldInfo[] classFields = myType.GetFields(BindingFlags.Static |
BindingFlags.Instance | BindingFlags.Public);
```




```
StringBuilder stringBuilder = new StringBuilder();

foreach (var field in classFields)
{
    stringBuilder.AppendLine($"{field.Name} must be private!");
}

MethodInfo[] setters = myType.GetMethods(BindingFlags.Instance |
BindingFlags.Public);
foreach (var setter in setters.Where(x => x.Name.StartsWith("set")))
{
    stringBuilder.AppendLine($"{setter.Name} have to be private");
}

MethodInfo[] getters = myType.GetMethods(BindingFlags.Instance |
BindingFlags.NonPublic);
foreach (var getter in getters.Where(x => x.Name.StartsWith("get")))
{
    stringBuilder.AppendLine($"{getter.Name} have to be public");
}

return stringBuilder.ToString();
}

internal class Program
{
    static void Main(string[] args)
    {
        Spy spy = new Spy();
        string newResult = spy.AnalyzeAccessModifiers("Hacker");
        Console.WriteLine(newResult);
    }
}
```

Задача 6.16. Мисия „Частно“ невъзможна

Време е да видим какво цели този хакер, с който се разправяме. Създайте метод в своя Spy клас, наречен `RevealPrivateMethods(string className)`. Отпечатайте всички частни методи в следния формат:

- All Private Methods of Class: {имеНаКлас}
- Base Class: {базовКлас}

На следващите редове отпечатайте имената на намерените методи, всеки на нов ред.

Използвайте `StringBuilder` да свържете отговора. Не променяйте нищо в "Hacker" класа!

В метода `Main` трябва да можете да проверите програмата си със следния код:



```
public static void Main()
{
    Spy spy = new Spy();
    string result = spy.RevealPrivateMethods("Hacker");
    Console.WriteLine(result);
}
```

Пример

Изход
All Private Methods of Class: Hacker Base Class: Object get_Id set_Id set_BankAccountBalance Finalize MemberwiseClone

Решение

```
namespace HackerMissionPrivate
{
    public class Hacker
    {
        public string username = "securityGod82";
        private string password = "mySuperSecretPassw0rd";

        public string Password
        {
            get => this.password;
            set => this.password = value;
        }

        private int Id { get; set; }

        public double BankAccountBalance { get; private set; }

        public void DownloadAllBankAccountsInTheWorld()
        {
            // nope
        }
    }

    public class Spy
    {
        public string RevealPrivateMethods(string className)
        {
            Type myType = Type.GetType(className);
            Console.WriteLine("All Private Methods of Class: {0}", className);
        }
    }
}
```



```
        Console.WriteLine("Base Class: {0}", myType.BaseType.Name);
        string fullname = myType.FullName;
        MethodInfo[] classFields = myType.GetMethods(BindingFlags.Instance |
BindingFlags.NonPublic);
        StringBuilder stringBuilder = new StringBuilder();
        foreach (var field in classFields)
        {
            stringBuilder.AppendLine($"{field.Name}");
        }
        return stringBuilder.ToString();
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Spy spy = new Spy();
        string result = spy.RevealPrivateMethods("Hacker");
        Console.WriteLine(result);
    }
}
```

Задача 6.17. Колектор

Използвайте отражение, за да уловите всички "Hacker" методи. След това подгответе алгоритъм, който да разпознае кои методи са getters и setters.

Отпечатайте на конзолата всеки getter на нов ред във формат:

- {име} will return {Връщан Tun}

След това отпечатайте всички setters във формат:

- {име} will set field of {Tun на параметър}

Използвайте StringBuilder да свържете отговора. Не променяйте нищо в "Hacker" класа!

В Main метода трябва да можете да проверите програмата си със следните няколко реда:

```
public static void Main()
{
    Spy spy = new Spy();
    string result = spy.CollectGettersAndSetters("Hacker");
    Console.WriteLine(result);
}
```



Пример

Изход

```
get_Password will return System.String  
get_Id will return System.Int32  
get_BankAccountBalance will return System.Double  
set_Password will set field of System.String  
set_Id will set field of System.Int32  
set_BankAccountBalance will set field of System.Double
```

Решение

```
namespace HackerCollector  
{  
    public class Hacker  
    {  
        public string username = "securityGod82";  
  
        private string password = "mySuperSecretPassw0rd";  
  
        public string Password  
        {  
            get => this.password;  
            set => this.password = value;  
        }  
  
        private int Id { get; set; }  
  
        public double BankAccountBalance { get; private set; }  
  
        public void DownloadAllBankAccountsInTheWorld()  
        {  
            // nope  
        }  
    }  
  
    public class Spy  
    {  
        public string CollectGettersAndSetters(string investigatedClass)  
        {  
            Type classType = Type.GetType(investigatedClass);  
  
            MethodInfo[] classMethods = classType.GetMethods(BindingFlags.Instance |  
BindingFlags.NonPublic | BindingFlags.Public);  
            StringBuilder stringBuilder = new StringBuilder();  
  
            foreach (MethodInfo method in classMethods.Where(m =>  
m.Name.StartsWith("get")))  
            {  
                stringBuilder.AppendLine($"{method.Name} will return  
{method.ReturnType}");  
            }  
        }  
    }  
}
```



```
        foreach (MethodInfo method in classMethods.Where(m =>
m.Name.StartsWith("set")))
        {
            stringBuilder.AppendLine($"{method.Name} will set field of
{method.ReturnType}");
        }

        return stringBuilder.ToString().Trim();
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Spy spy = new Spy();
        string result = spy.CollectGettersAndSetters("Hacker");
        Console.WriteLine(result);
    }
}
```

Задача 6.18. Поля за жътва

Даден ви е клас RichSoilLand с много полета (вижте предоставената конструкция). Като добър фермер, какъвто вие сте, трябва да ожънете тези полета. Да ги ожънете означава, че трябва да отпечатате всяко поле в определен формат (както е в изхода).

Вход

Ще получите максимум 100 реда с една от следните команди:

- private – отпечатва всички private полета
- protected – отпечатва всички protected полета
- public – отпечатва всички public полета
- all – отпечатва ВСИЧКИ декларирани полета
- HARVEST – край на входните данни

Изход

За всяка команда трябва да отпечатате полетата, които имат съответния модификатор за достъп, описан във входната секция. Форматът, в който полетата трябва да се отпечатат, е:

"<access modifier> <field type> <field name>"

Примери

Вход	Изход
protected HARVEST	protected String testString protected Double aDouble protected Byte testByte protected StringBuilder aBuffer



	protected BigInteger testBigNumber protected Single testFloat protected Object testPredicate protected Object fatherMotherObject protected String moarString protected Exception inheritableException protected Stream moarStreamz
private public private HARVEST	private Int32 testInt private Int64 testLong private Calendar aCalendar private Char testChar private BigInteger testBigInt private Thread aThread private Object aPredicate private Object hiddenObject private String anotherString private Exception internalException private Stream secretStream public Double testDouble public String aString public StringBuilder aBuilder public Int16 testShort public Byte aByte public Single aFloat public Thread testThread public Object anObject public Int32 anotherIntBitesTheDust public Exception justException public Stream aStream private Int32 testInt private Int64 testLong private Calendar aCalendar private Char testChar private BigInteger testBigInt private Thread aThread private Object aPredicate private Object hiddenObject private String anotherString private Exception internalException private Stream secretStream
all	private Int32 testInt



HARVEST	<pre>public Double testDouble protected String testString private Int64 testLong protected Double aDouble public String aString private Calendar aCalendar public StringBuilder aBuilder private Char testChar public Int16 testShort protected Byte testByte public Byte aByte protected StringBuilder aBuffer private BigInteger testBigInt protected BigInteger testBigNumber protected Single testFloat public Single aFloat private Thread aThread public Thread testThread private Object aPredicate protected Object testPredicate public Object anObject private Object hiddenObject protected Object fatherMotherObject private String anotherString protected String moarString public Int32 anotherIntBitesTheDust private Exception internalException protected Exception inheritableException public Exception justException public Stream aStream protected Stream moarStreamz private Stream secretStream</pre>
---------	--

Решение

```
namespace HarvestingFields
{
    public class HarvestingFields
    {
        private int testInt;
        public double testDouble;
        protected string testString;
        private long testLong;
        protected double aDouble;
        public string aString;
        private Calendar aCalendar;
```



```
public StringBuilder aBuilder;
private char testChar;
public short testShort;
protected byte testByte;
public byte aByte;
protected StringBuilder aBuffer;
private BigInteger testBigInt;
protected BigInteger testBigNumber;
protected float testFloat;
public float aFloat;
private Thread aThread;
public Thread testThread;
private object aPredicate;
protected object testPredicate;
public object anObject;
private object hiddenObject;
protected object fatherMotherObject;
private string anotherString;
protected string moarString;
public int anotherIntBitesTheDust;
private Exception internalException;
protected Exception inheritableException;
public Exception justException;
public Stream aStream;
protected Stream moarStreamz;
private Stream secretStream;
}

internal class Program
{
    static void Main(string[] args)
    {
        Type type = typeof(HarvestingFields);
        FieldInfo[] info = type.GetFields(BindingFlags.Public |
BindingFlags.Static | BindingFlags.Instance | BindingFlags.NonPublic);

        while (true)
        {
            string cmd = Console.ReadLine();
            switch (cmd)
            {
                case "HARVEST": return;
                case "protected":
                {
                    Console.WriteLine(string.Join("\n", info.Where(x =>
x.IsFamily).Select(x => $"protected {x.FieldType.Name} {x.Name}").ToArray()));
                    break;
                }
                case "private":
                {
                    Console.WriteLine(string.Join("\n", info.Where(x =>
x.IsPrivate).Select(x => $"private {x.FieldType.Name} {x.Name}").ToArray()));
                    break;
                }
                case "public":
```




```
        {
            Console.WriteLine(string.Join("\n", info.Where(x =>
x.IsPublic).Select(x => $"public {x.FieldType.Name} {x.Name}").ToArray()));
            break;
        }
        case "all":
        {
            Console.WriteLine(string.Join("\n", info.Select(x =>
$" {x.Attributes} {x.FieldType.Name} {x.Name}").ToArray()));
            break;
        }
    }
}
}
```

Задача 6.19. Black Box Integer

Помогате на свой приятел, който е все още в OOP Basics курса – името му е Пешослав (да не се бърка с реални хора или треньори). Той е малко „бавничък“ и е направил клас, в който всички членове са `private`. Вашите задачи са да представите с конкретни примери обект от неговия клас (винаги с начална стойност 0) и после да извикате всички методи, които той има. Ограничението ви е да не променяте „ръчно“ нищо в самия клас (смятайте го за черна кутия). Можете да разглеждате класа му, но не го пипайте! Самият клас се казва `BlackBoxInt` и е обвивка за базовия `int`.

Методите, които има този клас, са:

- `Add(int)`
- `Subtract(int)`
- `Multiply(int)`
- `Divide(int)`
- `LeftShift(int)`
- `RightShift(int)`

Вход

Входът ще се състои от редове във вида:

- `<име на метод>_<стойност>`

Например: `Add_115`

Входът винаги ще е валиден и в описания формат, така че няма нужда да го проверявате изрично. Спирате да получавате вход когато срещнете командата `"END"`.

Изход

Всяка команда (освен `END`) трябва да отпечата настоящата стойност на `innerValue` от `BlackBoxInt` обекта, който представяте. Не мамете, като



предефинирате ToString() в класа – трябва да вземете стойността от private полето.

Примери

Вход	Изход
Add_999999	999999
Subtract_19	999980
Divide_4	249995
Multiply_2	499990
RightShift_1	249995
LeftShift_3	1999960
END	

Решение

```
namespace BlackBoxInteger
{
    public class BlackBoxInteger
    {
        private static int DefaultValue = 0;

        private int innerValue;

        private BlackBoxInteger(int innerValue)
        {
            this.innerValue = innerValue;
        }

        private BlackBoxInteger()
        {
            this.innerValue = DefaultValue;
        }

        private void Add(int addend)
        {
            this.innerValue += addend;
        }

        private void Subtract(int subtrahend)
        {
            this.innerValue -= subtrahend;
        }

        private void Multiply(int multiplier)
        {
            this.innerValue *= multiplier;
        }

        private void Divide(int divider)
        {
            this.innerValue /= divider;
        }
    }
}
```



```
private void LeftShift(int shifter)
{
    this.innerValue <<= shifter;
}

private void RightShift(int shifter)
{
    this.innerValue >>= shifter;
}
}

internal class Program
{
    public static BlackBoxInteger instance;

    public static Type type;

    static void Main(string[] args)
    {
        // Fields
        type = typeof(BlackBoxInteger);
        // FieldInfo[] info = type.GetFields(BindingFlags.Public |
        BindingFlags.Static | BindingFlags.Instance | BindingFlags.NonPublic);

        instance = (BlackBoxInteger)Activator.CreateInstance(type, true);

        // Constructor
        // ConstructorInfo ctor = type.GetConstructor(new[] { typeof(int) });
        // object inst = ctor.Invoke(new object[] { 10 });

        // Methods
        MethodInfo[] methods = type.GetMethods(BindingFlags.Public |
        BindingFlags.Static | BindingFlags.Instance | BindingFlags.NonPublic);

        while (true)
        {
            string input = Console.ReadLine();
            if (input == "END") return;
            string cmd = input.Substring(0, input.IndexOf("_"));
            string argument = input.Substring(input.IndexOf("_") + 1);
            Console.WriteLine(MethodInvoker(cmd, argument));
        }

        public static string MethodInvoker(string cmd, string arguments)
        {
            MethodInfo singleMethod = type.GetMethod("Add", BindingFlags.NonPublic |
            BindingFlags.Instance);
            FieldInfo field = type.GetField("innerValue", BindingFlags.NonPublic |
            BindingFlags.Instance);
            // int field.SetValue(instance, 5);
            switch (cmd)
            {
                case "Add": break;
            }
        }
    }
}
```



```
        case "Subtract": singleMethod = type.GetMethod("Subtract",  
BindingFlags.NonPublic | BindingFlags.Instance); break;  
        case "Divide": singleMethod = type.GetMethod("Divide",  
BindingFlags.NonPublic | BindingFlags.Instance); break;  
        case "Multiply": singleMethod = type.GetMethod("Multiply",  
BindingFlags.NonPublic | BindingFlags.Instance); break;  
        case "RightShift": singleMethod = type.GetMethod("RightShift",  
BindingFlags.NonPublic | BindingFlags.Instance); break;  
        case "LeftShift": singleMethod = type.GetMethod("LeftShift",  
BindingFlags.NonPublic | BindingFlags.Instance); break;  
    }  
    var result = singleMethod.Invoke(instance, new object[] {  
int.Parse(arguments) });  
  
    return ((int)field.GetValue(instance)).ToString();  
}  
}
```

Задача 6.20. BarracksWars – Нова фабрика

Даден ви е малък конзолен проект, наречен Barracks (неговият код е включен в предоставената програмна конструкция).

Основните функционалности на проекта са добавяне на нови единици към склада и отпечатване на доклад със статистики за единиците, които в момента са в склада. Първо нека презгледаме оригиналната задача преди проектът да е бил създаден:

Вход

Входът се състои от команди, всяка на отделен ред. Командите, които изпълняват функционалностите, са:

- add <Archer/Swordsman/Pikeman/{...}> - добавя единица към склада.
- report – отпечатва статистика по лексикологичен ред за единиците в склада.
- fight – край на входните данни.

Изход

Всяка команда освен fight трябва да отпечатва изхода си на конзолата.

add трябва да отпечатва: "<Archer/Swordsman/Pikeman/{...}> added!"

report трябва да отпечатва цялата информация в склада във формата: "<UnitType> -> <UnitQuantity>", сортиран с UnitType

Ограничения

Входът ще представлява не повече от 1000 реда

Командата report никога няма да бъде дадена преди коя да е валидна команда да е дадена



Вашата задача

1) Трябва да проучите кода на проекта и да разберете как работи. В него обаче има части, които не са имплементирани (оставени са с TODO). Трябва да имплементирате функционалността на метода CreateUnit в класа UnitFactory, така че да създаде единица на базата на нейния тип, получен като параметър. Имплементирайте я по такъв начин, че когато добавите нова единица, тя да може да бъде създадена без да е необходимо да се променя нещо в UnitFactory класа (ще ви кажа на ушенце: използвайте отражение). Може да използвате подхода, наречен Simple Factory.

2) Добавете два нови класа за единици (ще има тестове, които ги изискват) - Horseman с 50 здраве и 10 атака и Gunner с 20 здраве и 20 атака.

Ако правилно изпълните всичко в тази задача, трябва да добавяте код само в namespace Factories и Units.

Примери

Вход	Изход
add Swordsman add Archer add Pikeman report add Pikeman add Pikeman report fight	Swordsman added! Archer added! Pikeman added! Archer -> 1 Pikeman -> 1 Swordsman -> 1 Pikeman added! Pikeman added! Archer -> 1 Pikeman -> 3 Swordsman -> 1
add Pikeman add Pikeman add Gunner add Horseman add Archer add Gunner add Gunner add Horseman report fight	Pikeman added! Pikeman added! Gunner added! Horseman added! Archer added! Gunner added! Gunner added! Horseman added! Archer -> 1 Gunner -> 3 Horseman -> 2 Pikeman -> 2

Решение

```
namespace BaracasWars  
{
```



```
public class Engine : IRunnable
{
    private IRepository repository;

    private IUnitFactory unitFactory;

    public Engine(IRepository repository, IUnitFactory unitFactory)
    {
        this.repository = repository;
        this.unitFactory = unitFactory;
    }

    public void Run()
    {
        while (true)
        {
            try
            {
                string input = Console.ReadLine();
                string[] data = input.Split();
                string commandName = data[0];
                string result = InterpretCommand(data, commandName);
                Console.WriteLine(result);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }

    private string InterpretCommand(string[] data, string commandName)
    {
        string result = string.Empty;
        switch (commandName)
        {
            case "add":
                result = this.AddUnitCommand(data);
                break;
            case "report":
                result = this.ReportCommand(data);
                break;
            case "fight":
                Environment.Exit(0);
                break;
            default:
                throw new InvalidOperationException("Invalid command!");
        }
        return result;
    }

    private string ReportCommand(string[] data)
    {
        string output = this.repository.Statistics;
```



```
        return output;
    }

    private string AddUnitCommand(string[] data)
    {
        string unitType = data[1];
        IUnit unitToAdd = this.unitFactory.CreateUnit(unitType);
        this.repository.AddUnit(unitToAdd);
        string output = unitType + " added!";
        return output;
    }
}

public interface IAttacker
{
    int AttackDamage { get; }
}

public interface ICommandInterpreter
{
    IExecutable InterpretCommand(string[] data, string commandName);
}

public interface IDestroyable
{
    int Health { get; set; }
}

public interface IExecutable
{
    string Execute();
}

public interface IRepository
{
    void AddUnit(IUnit unit);
    string Statistics { get; }
    void RemoveUnit(string unitType);
}

public interface IRunnable
{
    void Run();
}

public interface IUnit : IDestroyable, IAttacker
{
    // nope
}

public interface IUnitFactory
{
    IUnit CreateUnit(string unitType);
}
```



```
public class Archer : Unit
{
    private const int DefaultHealth = 25;

    private const int DefaultDamage = 7;

    public Archer() : base(DefaultHealth, DefaultDamage)
    {
        // nope
    }
}

public class Gunner : Unit
{
    private const int DefaultHealth = 20;

    private const int DefaultDamage = 20;

    public Gunner() : base(DefaultHealth, DefaultDamage)
    {
        // nope
    }
}

public class Horseman : Unit
{
    private const int DefaultHealth = 50;

    private const int DefaultDamage = 10;

    public Horseman() : base(DefaultHealth, DefaultDamage)
    {
        // nope
    }
}

public class Pikeman : Unit
{
    private const int DefaultHealth = 30;

    private const int DefaultDamage = 15;

    public Pikeman() : base(DefaultHealth, DefaultDamage)
    {
        // nope
    }
}

public class Swordsman : Unit
{
    private const int DefaultHealth = 40;

    private const int DefaultDamage = 13;

    public Swordsman() : base(DefaultHealth, DefaultDamage)
```




```
{
    // nope
}

public class Unit : IUnit
{
    private int health;

    private int attackDamage;

    protected Unit(int health, int attackDamage)
    {
        this.SetInitialHealth(health);
        this.AttackDamage = attackDamage;
    }

    public int AttackDamage
    {
        get
        {
            return this.attackDamage;
        }

        private set
        {
            if (value <= 0)
            {
                throw new ArgumentException("Attack damage should be
positive.");
            }

            this.attackDamage = value;
        }
    }

    public int Health
    {
        get
        {
            return this.health;
        }

        set
        {
            if (value < 0)
            {
                this.health = 0;
            }
            else
            {
                this.health = value;
            }
        }
    }
}
```



```
private void SetInitialHealth(int health)
{
    if (health <= 0)
    {
        throw new ArgumentException("Initial health should be positive.");
    }

    this.Health = health;
}

public class UnitFactory : IUnitFactory
{
    public IUnit CreateUnit(string unitType)
    {
        switch (unitType)
        {
            case "SwordsMan": return new Swordsman();
            case "Archer": return new Archer();
            case "Pikeman": return new Pikeman();
            case "Gunner": return new Gunner();
            case "Horseman": return new Horseman();
            default: return new Swordsman();
        }
    }
}

class UnitRepository : IRepository
{
    private IDictionary<string, int> amountOfUnits;

    public UnitRepository()
    {
        this.amountOfUnits = new SortedDictionary<string, int>();
    }

    public string Statistics
    {
        get
        {
            StringBuilder statBuilder = new StringBuilder();
            foreach (var entry in amountOfUnits)
            {
                string formattedEntry =
                    string.Format("{0} -> {1}", entry.Key, entry.Value);
                statBuilder.AppendLine(formattedEntry);
            }

            return statBuilder.ToString().Trim();
        }
    }

    public void AddUnit(IUnit unit)
    {

```



```
        string unitType = unit.GetType().Name;
        if (!this.amountOfUnits.ContainsKey(unitType))
        {
            this.amountOfUnits.Add(unitType, 0);
        }
        this.amountOfUnits[unitType]++;
    }

    public void RemoveUnit(string unitType)
    {
        throw new NotImplementedException();
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        IRepository repository = new UnitRepository();
        IUnitFactory unitFactory = new UnitFactory();
        IRunnable engine = new Engine(repository, unitFactory);
        engine.Run();
    }
}
```

Задача 6.21. BarracksWars – Командите отвържат на удара

Както може би сте забелязали, командите в проекта от задача 3 са имплементирани чрез a switch case и извиквания на методи в класа Engine. Въпреки че този подход работи, той има недостатъци когато добавяте нова команда, защото за нея трябва да бъде добавен нов case. В някои проекти може да нямате достъп до класа Engine и това не би работило. Представете си този проект да е outsource-нат - outsourcing фирмата няма да има достъп до двигателя. Направете така, че когато искат да добавят нова команда, нищо в Engine да не трябва да се променя.

За да постигнете това, употребете шаблон в проектирането, наречен Command Pattern. Правила сме това и преди в BashSoft Lab и можете да погледнете и там за съвети. Използвайте предоставения интерфейс Executable като рамка за класовете на командите. Поставете новите командни класове в предоставения commands пакет в core. Също може да направите интерпретатор на команди, за да откачите тази функционалност от Engine. Ето как трябва да изглежда базовата (абстрактна) команда:



```
public abstract class Command : IExecutable
{
    private string[] data;
    private IRepository repository;
    private IUnitFactory unitFactory;

    protected Command(string[] data, IRepository repository, IUnitFactory unitFactory)
    {
        this.Data = data;
        this.Repository = repository;
        this.UnitFactory = unitFactory;
    }

    protected string[] Data
    {
        get { return this.data; }
        private set { this.data = value; }
    }

    protected IRepository Repository
    {
        get{ return this.repository; }
        private set{ this.repository = value; }
    }

    protected IUnitFactory UnitFactory
    {
        get { return this.unitFactory; }
        private set { this.unitFactory = value; }
    }

    public abstract string Execute();
}
```

Забележете, че всички команди, които разширяват тази, ще имат както склад, така и UnitFactory, въпреки че не всички се нуждаят от тях. Оставете това така за тази задача, защото за да работи отражението трябва всички конструктори да приемат едни и същи параметри. Ще видим как да заобиколим този проблем в следващата задача.

След като сте имплементирали шаблона, добавете нова команда. Ще има следният синтаксис:

retire <UnitType> - Всичко, което трябва да направи, е да премахне единица от дадения тип от склада.

Ако в момента няма такива единици в склада, отпечатайте: "No such units in repository."



Ако в момента има такъв единици в склада, отпечатайте: "<UnitType> retired!"

За да имплементирате командата, ще трябва да имплементирате и съответния метод в UnitRepository.

Ако правилно изпълните всичко в тази задача, трябва да пишете/променяте код само в Core и Data пакетите.

Примери

Вход	Изход
retire Archer	No such units in repository.
add Pikeman	Pikeman added!
add Pikeman	Pikeman added!
add Gunner	Gunner added!
add Horseman	Horseman added!
add Archer	Archer added!
add Gunner	Gunner added!
add Gunner	Gunner added!
add Horseman	Horseman added!
report	Archer -> 1
retire Gunner	Gunner -> 3
retire Archer	Horseman -> 2
report	Pikeman -> 2
retire Swordsman	Gunner retired!
retire Archer	Archer retired!
fight	Archer -> 0
	Gunner -> 2
	Horseman -> 2
	Pikeman -> 2
	No such units in repository.
	No such units in repository.

Решение

Задача 6.22. * BarracksWars - Завръщане на Зависимостите

В последната част от тази епична трилогия от задачи ще разрешим проблема, при който всички команди получават всички utility класове като параметри в своите конструктори. Можем да постигнем това, използвайки подход, наречен dependency injection container. Този подход се използва в много библиотеки.

Ще променим малко този подход. Премахнете всички полета от абстрактните команди освен data. Вместо това сложете каквито полета са нужни на всяка команда в конкретния клас. Създайте параметър,



наречен Inject, и направете така, че да може да се използва само на полета. Подайте този параметър за полетата, които трябва да променим чрез отражение. След като сте подготвили всичко това, напишете необходимия код за отражение в Командния Интерпретатор (който трябва да сте преработили от Engine в предната задача).

Можете да използвате същия пример като в предната задача, за да проверите дали сте я изпълнили правилно.

Решение

Тема 7. Елементи от функционалното програмиране

Задача 7.1. Сортиране на нечетни числа

Напишете програма, която въвежда цели числа разделени с ",". Изведете нечетните числа в множеството, сортирани в нарастващ ред..

Примери

Вход	Изход	Вход	Изход	Вход	Изход
4, 2, 1, 3, 5, 7, 1, 4, 2, 12	2, 2, 4, 4, 12	1, 3, 5		2, 4, 6	2, 4, 6

Подсказки

Изберете какъв тип на структури от данни ще използвате за решаване на този проблем. Използвайте функционален програмен филтър и сортирайте колекцията от числа.

Решение

```
namespace SortEvenNumbers
```

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            int[] numbers = Console.ReadLine()  
                .Split(new string[] { "," },  
StringSplitOptions.RemoveEmptyEntries)  
                .Select(n => int.Parse(n))  
                .Where(n => n % 2 == 0)  
                .OrderBy(n => n)  
                .ToArray();  
            Console.WriteLine(string.Join(", ", numbers));  
        }  
    }  
}
```

Задача 7.2. Сбор на числа

Напишете програма, която чете редица от цели числа, разделени с ",". Извежда на два реда на броя на числата и тяхната сума.



Примери

Вход	Изход
4, 2, 1, 3, 5, 7, 1, 4, 2, 12	10 41
2, 4, 6	3 12

Решение

`namespace SumNumbers`

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Func<int[]> input = () => Console.ReadLine()  
                .Split(new string[] { ", " },  
StringSplitOptions.RemoveEmptyEntries)  
                .Select(int.Parse).ToArray();  
            int[] numbers = input();  
            Func<int[], int> count = (nums) => nums.Count();  
            Console.WriteLine(count(numbers));  
            Func<int[], int> sum = (nums) => nums.Sum();  
            Console.WriteLine(sum(numbers));  
        }  
    }  
}
```

Задача 7.3. Брой думи главна буква

Напишете програма, която чете текст на един ред от конзолата. Извежда на всички думи, които започват с главна буква в същия ред, в който сте ги получили в текста.

Примери

Вход	Изход
The following example shows how to use Function	The Function
Write a program that reads one line of text from console. Print count of words that start with Uppercase, after that print all those words in the same order like you find them in text.	Write Print Uppercase,

Подсказки

- Използвайте `Func<string, bool>` както в if условие
- Използвайте „“ за разделител.

Решение

`namespace CountUppercaseWords`



```
{
    internal class Program
    {
        static void Main(string[] args)
        {
            var words = Console.ReadLine()
                .Split(new string[] { " " },
                    StringSplitOptions.RemoveEmptyEntries);
            Func<string, bool> checker = n => n[0] == n.ToUpper()[0];
            words.Where(checker)
                .ToList()
                .ForEach(n => Console.WriteLine(n));
        }
    }
}
```

Задача 7.4. Начисляване на ДДС (VAT)

Напишете програма, която чете един ред на дробни числа за цени, разделени с ",". Извежда цените с добавен ДДС за всички тях. Форматирайте ги на 2 знака след десетичната точка. Редът на цените трябва да бъде същия. ДДС е равна на 20 % от цената.

Примери

Вход	Изход
1.38, 2.56, 4.4	1.66 3.07 5.28

Вход	Изход
1, 3, 5, 7	1.20 3.60 6.00 8.40

Решение

```
namespace AddVAT
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.ReadLine()
                .Split(new string[] { ",", " " },
                    StringSplitOptions.RemoveEmptyEntries)
                .Select(double.Parse)
                .Select(n => n * 1.2)
                .ToList()
                .ForEach(n => Console.WriteLine($"{n:F2}"));
        }
    }
}
```

Задача 7.5. Филтриране по възраст

Напишете програма, която получава цяло число N на първия ред. На следващите N реда, въведете двойки от "[име], [възраст]". След това извежда три реда със:



- състояние - "по-млад" или "по-стар"
- възраст – цяло число
- формат - "име", "възраст" или "име възраст"

в зависимост от условието, изведете правилната двойка в правилния формат. Не използвайте вградената функционалност от .NET. Създайте собствени методи.

Примери

Вход	Изход	Вход	Изход	Вход	Изход
5	Pesho - 20	5	Gosho	5	20
Pesho, 20	Mimi - 29	Pesho, 20	Simo	Pesho, 20	18
Gosho, 18	Ico - 31	Gosho, 18		Gosho, 18	29
Mimi, 29		Mimi, 29		Mimi, 29	31
Ico, 31		Ico, 31		Ico, 31	16
Simo, 16		Simo, 16		Simo, 16	
older		younger		younger	
20		20		50	
name age		name		age	

Решение

```
namespace FilterByAge
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<string, int> people = new Dictionary<string, int>();

            int n = int.Parse(Console.ReadLine());
            for (int i = 0; i < n; i++)
            {
                var input = Console.ReadLine().Split(new string[] { ", " },
                    StringSplitOptions.RemoveEmptyEntries);
                people.Add(input[0], int.Parse(input[1]));
            }
            string condition = Console.ReadLine();
            int age = int.Parse(Console.ReadLine());
            string format = Console.ReadLine();

            Func<int, bool> tester = CreateTester(condition, age);
            Action<KeyValuePair<string, int>> printer = CreatePrinter(format);
            PrintFilteredStudent(people, tester, printer);
        }

        public static Func<int, bool> CreateTester(string condition, int age)
        {
            switch (condition)
            {
                case "younger": return x => x < age;
                case "older": return x => x >= age;
            }
        }
    }
}
```



```
        default: return null;
    }
}

public static Action<KeyValuePair<string, int>> CreatePrinter(string format)
{
    switch (format)
    {
        case "name":
            return person => Console.WriteLine($"{person.Key}");
        case "age":
            return person => Console.WriteLine($"{person.Value}");
        case "name age":
            return person =>
                Console.WriteLine($"{person.Key} - {person.Value}");
        default: return null;
    }
}

public static void PrintFilteredStudent
(
    Dictionary<string, int> people,
    Func<int, bool> tester,
    Action<KeyValuePair<string, int>> printer
)
{
    foreach (var person in people)
    {
        if (tester(person.Value))
        {
            printer(person);
        }
    }
}
}
```

Задача 7.6. Action

Напишете програма, която чете колекция от низове от конзолата и след това ги отпечата на конзолата. Всяко име трябва да бъде отпечатано на нов ред. Използвайте Action<T>

Примери

Вход	Изход
Pesho Gosho Adasha	Pesho Gosho Adasha

Решение

```
namespace Action
{
    class Program
    {
```



```
static void Main(string[] args)
{
    string[] input = Console.ReadLine().Split().ToArray();

    Action<string[]> print = x => {
        foreach (string item in x)
            Console.WriteLine(item);
    };

    print(input);
}
```

Задача 7.7. Рицари на честта

Напишете програма, която чете колекция от имена като символни низове от конзолата след това добавя "Sir" пред всяко име и го отпечатва обратно върху конзолата. Използвайте Action<T>

Примери

Вход	Изход
Pesho Gosho Adasha StanleyRoyce	Sir Pesho Sir Gosho Sir Adasha Sir StanleyRoyce

Решение

```
namespace KnightsOfHonor
{
    class Program
    {
        static void Main(string[] args)
        {
            Func<string[]> input = () => Console.ReadLine().Split(new[] { ' ' },
                StringSplitOptions.RemoveEmptyEntries).ToArray();
            Action<string[]> output = (inp) => Console.WriteLine("Sir " +
                string.Join("\nSir ", inp));
            output(input());
        }
    }
}
```

Задача 7.8. Собствена Min функция

Напишете една проста програма, която чете от конзолата набор от цели числа и отпечатва обратно върху конзолата най-малкото число от колекцията. Използвайте Func<T, T>.

Примери

Вход	Изход
1 4 3 2 1 7 13	1



Решение

```
namespace MinFunction
{
    class Program
    {
        static void Main(string[] args)
        {
            static void Main(string[] args)
            {
                Func<int[]> input = () => Console.ReadLine().Split(new[] { ' ' },
StringSplitOptions.RemoveEmptyEntries).Select(int.Parse).ToArray();
                Action<int[]> min = (inp) => Console.WriteLine(inp.Min());
                min(input());
            }
        }
    }
}
```

Задача 7.9. Намерете четно или нечетно

Дадени са Ви долна и горна граница за интервал от число. След това команда определя, ако трябва да се изброят всички четни и нечетни числа в дадения интервал. Използвайте Predicate<T>

Примери

Вход	Изход
1 10 odd	1 3 5 7 9
20 30 even	20 2 24 26 28 30

Решение

```
namespace OddOrEven
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int[] numbers = Console.ReadLine().Split().Select(int.Parse).ToArray();
            string oddOrEven = Console.ReadLine();

            Predicate<int> evenFinder = (int p) => { return p % 2 == 0; };
            Predicate<int> oddFinder = (int p) => { return p % 2 != 0; };
            int[] result = null;

            if (oddOrEven == "odd")
            {
                result = Enumerable.Range(numbers[0], numbers[1] - numbers[0] +
1).Where(n => oddFinder(n)).ToArray();
            }
            else if (oddOrEven == "even")
            {
            }
```



```
        result = Enumerable.Range(numbers[0], numbers[1] - numbers[0] +  
1).Where(n => evenFinder(n)).ToArray();  
    }  
  
    Console.WriteLine(string.Join(" ", result));  
}  
}
```

Задача 7.10. Приложна аритметика

Напишете програма, която изпълнява някои математически операции върху дадена колекция. На първия ред са дадени списък от числа. На следващите редове се подават различни команди, които трябва да се прилагат за всички числа в списъка: "Add"-> добави 1 към всяко число; "multiply"-> умножава всяко число по 2; "subtract"-> изважда 1 от всяко число; "print"-> извежда колекцията. Входът ще приключи с команда "end". Използвайте функции.

Примери

Вход	Изход
1 2 3 4 5 add add print end	3 4 5 6 7
5 10 multiply subtract print end	9 19

Решение

```
namespace Aritmetics  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            int[] numbers = Console.ReadLine().Split().Select(int.Parse).ToArray();  
            string command = Console.ReadLine();  
  
            while (command != "end")  
            {  
                switch (command)  
                {  
                    case "add":  
                        numbers = numbers.Select(obj => obj += 1).ToArray();  
                        break;  
                    case "print":  
                        Console.WriteLine(string.Join(" ", numbers));  
                }  
                command = Console.ReadLine();  
            }  
        }  
    }  
}
```



```
        break;
    case "multiply":
        numbers = numbers.Select(obj => obj *= 2).ToArray();
        break;
    case "subtract":
        numbers = numbers.Select(obj => obj -= 1).ToArray();
        break;
    }

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

Задача 7.11. Обърни и изпълни

Напишете програма, която обръща колекция и премахва елементи, които са се дели на дадено число n. Използвайте предикати/функции.

Примери

Вход	Изход
1 2 3 4 5 6 2	5 3 1
20 0 40 30 60 50 3	50 0 10 20

Решение

```
namespace ReverseAndExecute
{
    class Program
    {
        static void Main(string[] args)
        {
            List<int> numbers =
            Console.ReadLine().Split().Select(int.Parse).ToList();
            int divider = int.Parse(Console.ReadLine());

            Func<int, bool> checker = n => n % divider != 0;

            numbers = numbers.Where(checker).ToList();
            numbers.Reverse();

            Console.WriteLine(string.Join(" ", numbers));
        }
    }
}
```

Задача 7.12. Предикати за имена

Напишете програма, която филтрира списък с имена по тяхната дължина. На първия ред ще ви бъде дадено цяло число n, представляващо дължина на име. На втория ред ще ви бъде дадени някои имена като символни низове,



разделени с интервал. Напишете функция, която отпечатва само имената, чиято дължина е по-малка или равна на n.

Примери

Вход	Изход
4 Kurnelia Qnaki Geo Muk Ivan	Geo Muk Ivan
4 Karaman Asen Kiril Yordan	Asen

Решение

```
namespace NamePredicates
{
    class Program
    {
        static void Main(string[] args)
        {
            int n = int.Parse(Console.ReadLine());
            string[] ime = Console.ReadLine().Split(' ').ToArray();
            Console.WriteLine(string.Join("\n", ime.Where(x => x.Length <= n)));
        }
    }
}
```

Задача 7.13. Потребителски сравнител

Напишете потребителски компаратор, който сортира всички четни числа преди всички нечетни такива във възходящ ред. Подайте го на функция `Array.Sort()` и изведе резултата. Използвайте функции.

Примери

Вход	Изход
1 2 3 4 5 6	2 4 6 1 3 5
-3 2	2 -3

Решение

```
namespace UserComparator
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int[] ime = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();
            int[] even = ime.Where(x => x % 2 == 0).OrderBy(x => x).ToArray();
            int[] odd = ime.Where(x => x % 2 != 0).OrderBy(x => x).ToArray();
            Console.WriteLine(string.Join(" ", even.Concat(odd)));
        }
    }
}
```



Задача 7.14. Списък от предикати

Намерете всички числа в диапазона 1... N, които са делими от числата на определена последователност. На първия ред ще ви бъде дадена цяло число N –, което е края на интервала. На втория ред ще ви бъде дадена последователност от числа, които са делители. Използвайте предикати/функции.

Примери

Вход	Изход
10 1 1 1 2	2 4 6 8 10
100 2 5 10 20	20 40 60 80 100

Решение

```
namespace PredicatesList
```

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            int N = int.Parse(Console.ReadLine());  
            int[] dividers = Console.ReadLine().Split(''  
'').Select(int.Parse).ToArray();  
            for (int i = 1; i <= N; i++)  
            {  
                bool isDiv = true;  
                foreach (var item in dividers)  
                {  
                    if (i % item != 0) { isDiv = false; break; }  
                }  
                if (isDiv)  
                    Console.Write(i + " ");  
            }  
        }  
    }  
}
```

Задача 7.15. Предикатно парти!

Родителите на Иванчо са на почивка за празниците и той планира епично парти у дома. За съжаление неговите организационни умения са почти никакви и така ви се предоставя задачата да му помогнете с резервации. На първия ред получавате списък с всички хора, които ще дойдат. На следващите редове докато получите командата "Party!", може да бъдете помолени да удвоите или премахнете всички хора, които удовлетворяват дадени критерии. Има три различни критерии: 1. всеки, чието име започва с даден низ; 2. всеки, чието име завършва с даден низ; 3. всеки, чието име е с дадена дължина. Накрая отпечатвате всички гости, които ще са на партито, разделени с "," и след това добавяте окончанието "are going to the



party!". Ако няма никакви гости, които да отидат на партито отпечатайте "Nobody is going to the party!". Вижте примерите по-долу:

Примери

Вход	Изход
Pesho Misho Stefan Remove StartsWith P Double Length 5 Party!	Misho, Misho, Stefan are going to the party!
Pesho Double StartsWith Pesh Double EndsWith esho Party!	Pesho, Pesho, Pesho, Pesho are going to the party!
Pesho Remove StartsWith P Party!	Nobody is going to the party!

Решение

```
namespace PredicateParty
{
    class Program
    {
        static void Main(string[] args)
        {
            var people = Console.ReadLine().Split().ToList();
            Action<string> dabul = name => people.Add(name);
            Action<string> remove = name => people.Remove(name);
            Func<string, List<string>> contain = cont => { return people.Where(x =>
x.Contains(cont)).ToList(); };
            Func<int, List<string>> lenght = len => { return people.Where(x =>
x.Length == len).ToList(); };
            while (true)
            {
                var cmd = Console.ReadLine().Split().ToArray();
                switch (cmd[0])
                {
                    case "Party!":
                        if (people.Count != 0) Console.WriteLine($"{string.Join(",
", people)} are going to the party!");
                        else Console.WriteLine("Nobody is going to the party!");
                        return;
                    case "Remove":
                        if (cmd[1] == "Length ") lenght(int.Parse(cmd[2])).ForEach(x
=> remove(x));
                        else contain(cmd[2]).ForEach(x => remove(x));
                        break;
                    case "Double":
                        if (cmd[1] == "Length ")
                            lenght(int.Parse(cmd[2])).ForEach(x => dabul(x));
                        else contain(cmd[2]).ForEach(x => dabul(x));
                        break;
                }
            }
        }
    }
}
```



```
    }  
  }  
}
```

Задача 7.16. Модул Филтър за резервация на партита

Вие трябва да реализирате филтриращ модул за софтуер за резервации на партита. Първо, на модула Филтър за резервация на партита (PRFM за кратко) е подаден списък с покани. После PRFM получава последователност от команди, които указват дали е необходимо да добавите или премахнете даден филтър. Всички команди към PRFM се във формат {команда; тип на филтъра; филтриращ параметър} можете да получите следните PRFM команди: "Add filter", "Remove filter" или "Print". Всички PRFM филтърни типове са: "Starts with", "Ends with", "Length" и "Contains". Всички PRFM филтър параметри ще бъдат низове (или цяло число само за филтъра "Length"). Всяка команда ще бъде валидна например вие няма да бъдете помолени да премахнете несъществуващ филтър. Входът ще завърши с команда "Print", след което трябва да отпечатате всички, които отиват на партита, останали след филтриране. Вижте примерите по-долу:

Примери

Вход	Изход
Pesho Misho Slav Add filter;Starts with;P Add filter;Starts with;M Print	Slav
Pesho Misho Jica Add filter;Starts with;P Add filter;Starts with;M Remove filter;Starts with;M Print	Misho Jica

Решение

```
namespace PredicatePartyFilter  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            var people = Console.ReadLine().Split(new[] { ";" },  
StringSplitOptions.RemoveEmptyEntries).ToList();  
            var invited = new List<string>();  
            Action<string> add = name => invited.Add(name);  
            Action<string> remove = name => invited.Remove(name);  
            Func<string, List<string>> contain = cont => { return people.Where(x =>  
x.Contains(cont)).ToList(); };  
            Func<int, List<string>> lenght = len => { return people.Where(x =>  
x.Length == len).ToList(); };  
        }  
    }  
}
```



```
while (true)
{
    var cmd = Console.ReadLine().Split().ToArray();
    switch (cmd[0])
    {
        case "Add filter":
            if (cmd[1] == "Length ") lenght(int.Parse(cmd[2])).ForEach(x
=> add(x));
            else contain(cmd[2]).ForEach(x => add(x));
            break;
        case "Remove filter":
            if (cmd[1] == "Length ") lenght(int.Parse(cmd[2])).ForEach(x
=> remove(x));
            else contain(cmd[2]).ForEach(x => remove(x));
            break;
        case "Print":
            Console.WriteLine(string.Join(" ", invited));
            return;
    }
}
```

Задача 7.17. Inferno III.

На първия ред са дадени поредица от числа. Всяко число е скъпоценност и стойността представлява силата му. На следващите редове, докато получите командата Forge, ще получавате команди в следния формат: {команда; тип филтър; филтриращ параметър }. Командите могат да бъдат: Exclude, Reverse или Forge. Възможните типове Филтър са: Sum Left, Sum Right и Sum Left Right. Всички филтриращи параметри ще бъдат цяло число.

- Exclude маркира скъпоценност за изключване от множеството, ако тя отговаря на дадени условия,
- Reverse изтрива предишното изключване.
- Sum Left проверява дали силата на скъпоценния камък, добавен към скъпоценните камъни, стоящи отляво, дава определена стойност.
- Sum Right е същото, но за дясната страна на скъпоценния камък.
- Sum Left Right сумира силата на скъпоценния камък с левия и десния съсед.

Ако скъпоценен камък няма съсед отдясно или отляво (първия или последния елемент), тогава просто се добавя 0 за скъпоценност.

Имайте предвид, че промени в поредицата се прилагат само след коване Forge. Това означава, че скъпоценностите са фиксирани на техните позиции и всяка функция се случва на оригиналния набор, така че всяка скъпоценност се взема предвид, без значение дали е маркирана да бъде изключена или не. За по-добро разбиране на проблема, вижте примерите по-долу:



Примери

Вход	Изход	Коментари
1 2 3 4 5 Exclude;Sum Left;1 Exclude;Sum Left Right;9 Forge	2 4	1. Marks for exclusion all gems for which the sum with neighbors to their left equals 1, e.g. $0 + 1 = 1$ 2. Marks for exclusion all gems for which the sum with neighbors to their left and their right equals 9, e.g. $2 + 3 + 4 = 9$ $4 + 5 + 0 = 9$
1 2 3 4 5 Exclude;Sum Left;1 Reverse;Sum Left;1 Forge	1 2 3 4 5	1. Marks for exclusion all gems for which the sum with their gem peers to the left equals 1, e.g. $0 + 1 = 1$ 2. Reverses the previous exclusion.

Решение

```
namespace InfernoIII
{
    class Program
    {
        static void Main(string[] args)
        {
            var stones = Console.ReadLine().Split(new[] { " " },
StringSplitOptions.RemoveEmptyEntries).Select(int.Parse).ToList();
            int initialCount = stones.Count();
            var lastExcluded = new List<int>();
            Action<int> add = (s) =>
            {
                stones.AddRange(lastExcluded); lastExcluded = new List<int>();
            };
            Action<int> remove = name =>
            {
                stones.Remove(name); lastExcluded.Add(name);
            };
            Func<int, List<int>> containLR = s =>
            {
                List<int> result = new List<int>();
                if (s == 3) result.Add(1);
                else if (s == initialCount * 2 - 1) result.Add(initialCount);
                result.AddRange(stones.Where(x => 3 * x == s).ToList());
                return result;
            };
            Func<int, List<int>> containL = s =>
            {
```



```
List<int> result = new List<int>();
if (s == 1) result.Add(1);
result.AddRange(stones.Where(x => 2 * x - 1 == s).ToList());
return result;
};
Func<int, List<int>> containR = s =>
{
    List<int> result = new List<int>();
    if (s == initialCount) result.Add(initialCount);
    result.AddRange(stones.Where(x => 2 * x + 1 == s).ToList());
    return result;
};
while (true)
{
    var cmd = Console.ReadLine().Split(new[] { ";" },
StringSplitOptions.RemoveEmptyEntries).ToArray();
    switch (cmd[0])
    {
        case "Reverse":
            add(0);
            break;
        case "Exclude":
            if (cmd[1] == "Sum Left Right")
                containLR(int.Parse(cmd[2])).ForEach(x => remove(x));
            else if (cmd[1] == "Sum Left")
                containL(int.Parse(cmd[2])).ForEach(x => remove(x));
            else containR(int.Parse(cmd[2])).ForEach(x => remove(x));
            break;
        case "Forge":
            Console.WriteLine(string.Join(" ", stones.OrderBy(x =>
x).Distinct())));
            return;
    }
}
}
```

Задача 7.18. TriFunction

Напишете програма, която получава колекция от имена и връща първото име, чийто сбор от символи е равен на или по-голям от дадено число N, което ще бъде изведено на един ред. Използвайте функция, която приема друга функция като един от нейните параметри. Започнете с изграждането на регулярна функция, която да държи основната логика на програмата. Нещо по `Func<string, int, bool>`. След това създайте главна функция, която трябва да приеме първата функция като един от нейните параметри.

Примери

Вход	Изход
800	Petromir



Qvor Qnaki Petromir Sadam	
---------------------------	--

Решение

```
namespace TriFunction
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

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

```
            Func<string[]> input = () => Console.ReadLine()
```

```
                .Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)
```

```
                .ToArray();
```

```
            Func<string, bool> isBigger = (input) => input.ToCharArray().Sum(x => x)
```

```
>= 800;
```

```
            Action<string[]> output = (inp) =>
```

```
            {
```

```
                foreach (var item in inp)
```

```
                {
```

```
                    if (isBigger(item))
```

```
                        Console.WriteLine(item);
```

```
                };
```

```
            };
```

```
            output(input());
```

```
        }
```

```
    }
```

```
}
```

Тема 8. Комуникация между обекти

Задача 8.1. Реализация на събитие

Създайте клас Dispatcher със свойство name и клас Handler. Създайте публичен делегат, наречен NameChangeEventHandler, връщащ `void` в именното пространство (namespace) на Dispatcher (Но вън от класа Dispatcher) и събитие (поле от типа на делегата) вътре в класа Dispatcher с име NameChange. Създайте клас NameChangeEventArgs, който наследява класа EventArgs и има свойство – name, което се получава през конструктора и има private сетър (setter) и public гетър (getter). Създайте също и метод, наречен OnNameChange(NameChangeEventArgs args) в Dispatcher – това е метод, който ще бъде извикван да „запали“ събитието. В сетъра за име на Dispatchera, извикайте метода OnNameChange и го предайте на обект от тип NameChangeEventArgs с новата стойност на името, предадено в сетъра.

Напишете метод OnDispatcherNameChange(object sender, NameChangeEventArgs args) в класа Handler, реализацията трябва да извежда на конзолата "Dispatcher's name changed to <newName>". При стартиране на програмата, създайте нов Dispatcher и Handler, могава добавете метод OnDispatcherNameChange на бработчика (Handler) към събитието NameChange в класа Dispatcher.



Вход

От конзолата ще получите редове, съдържащи имена, докато получите командата "Край". За всяка промяна на име, се сменя името на диспечера с това. Винаги когато името на диспечера се променя, вие трябва да запалите събитие за всички наблюдатели (слушатели, абонати).

Изход

при всяка промяна на име на диспечера, обработчика да извежда на екрана "Dispatcher's name changed to <newName>."

Ограничения

- Имената да съдържат само букви.
- Броят команди да е цяло число в интервала [1...100].
- Последната команда винаги да е команда "End".

Примери

Вход	Изход
Pesho Gosho Stefan End	Dispatcher's name changed to Pesho. Dispatcher's name changed to Gosho. Dispatcher's name changed to Stefan.
Prakash Stamat MuadDib Ivan Joro End	Dispatcher's name changed to Prakash. Dispatcher's name changed to Stamat. Dispatcher's name changed to MuadDib. Dispatcher's name changed to Ivan. Dispatcher's name changed to Joro.

Решение

namespace NameChangeEvent

```
{  
    class NameChangeEventArgs : EventArgs  
    {  
        private string name;  
  
        public string Name  
        {  
            get { return name; }  
            private set { name = value; }  
        }  
  
        public NameChangeEventArgs(string n)  
        {  
            name = n;  
        }  
    }  
  
    class Handler  
    {  
        public void OnDispatcherNameChange(object sender, NameChangeEventArgs args)
```



```
{
    Console.WriteLine("Dispatcher's name changed to " + args.Name);
}

public delegate void NameChangeEventHandler();

class Dispatcher
{
    private string name;
    private event NameChangeEventHandler NameChange;

    public string Name
    {
        get { return name; }
        set
        {
            name = value;
            OnNameChange(new NameChangeEventArgs(name));
        }
    }

    public void OnNameChange(NameChangeEventArgs args)
    {
        Handler handler = new Handler();
        handler?.OnDispatcherNameChange(this, args);
    }
}

class Program
{
    static void Main(string[] args)
    {
        Dispatcher dp = new Dispatcher();
        string input = Console.ReadLine();
        while (input != "End")
        {
            dp.Name = input;
            input = Console.ReadLine();
        }
    }
}
```

Задача 8.2. Царски гамбит

Направете 3 класа - King, Footman и Royal Guard. Всички имат name (имената са уникални. Няма две единици с едно и също име), Слугите и царските охранители може също да бъдат убивани (убитите се премахват от програмата), докато е възможно да се атакува царя – трябва да имате метод за отговор на атаките. Когато царят атакуван, той трябва да извежда н конзолата "King <kingName> is under attack!" и всички живи слуги и царски пазачи трябва да отговорят на атаката:



- Слугата отговаря, като извежда на екрана "Footman <footmanName> is panicking!".
- Царският охранител извежда. Вместо това "Royal Guard <guardName> is defending!".

Вход

В първия ред на конзолата ще получите един низ - името на царя. На втория ред ще получите имената на неговите царски охранители, разделени с интервали. На третия имената на неговите слуги, разделени с интервали. На следващите редове, докато се получи командата "Край", вие ще получите команди в следния формат:

- "Attack King" – извиква отговор на царя към атакуващия.
- "Kill <name>" – слуга или охранител с даденото име да се убие

Изход

Когато царя се атакува трябва да изведете на конзолата "King <kingName> is under attack!" и всеки жив слуга и охранител (пазач) извежда тяхното съобщение за отговор – първо всички пазачи трябва да отговорят (в реда, който с въведени) и тогава слугите в същия ред. Всяко съобщение се извежда на нов ред.

Ограничения

- Имената ще съдържат само букви.
- Да им винаги един цар и поне един слуга и един охранител.
- Царят не се убива – няма команда за убиване на царя.
- Командите за убиване се получават само за живи войни
- Всички команди се получават в посочения формат
- Броят на командите ще е положително цяло число между [1...100].
- Последната команда винаги е "End".

Примери

Вход	Изход
Pesho Krivogled Ruboglav Gosho Pencho Stamat Attack King End	King Pesho is under attack! Royal Guard Krivogled is defending! Royal Guard Ruboglav is defending! Footman Gosho is panicking! Footman Pencho is panicking! Footman Stamat is panicking!
HenryVIII Thomas Oliver Mark Kill Oliver Attack King Kill Thomas	King HenryVIII is under attack! Royal Guard Thomas is defending! Footman Mark is panicking! King HenryVIII is under attack!



Kill Mark Attack King End	
---------------------------------	--

Решение

```
namespace KingGambit
{
    class Attacker
    {
        public delegate void RoyalGuardHandler(object sender, EventArgs e);

        public event RoyalGuardHandler WorkPerformed;

        public virtual void DoWork(object sender, EventArgs e)
        {
            WorkPerformed?.Invoke(sender, e);
            /* workPerformed(e);
            if (e is RoyalGuard)
            {
                RoyalGuard rg = ;
                WorkPerformed?.Invoke((RoyalGuard)e);
            }
            else if(e is Footman)
            {
            }
            */
        }

        public void workPerformed(object sender, EventArgs e)
        {
            // if (e is RoyalGuard)
            // Console.WriteLine("Finished after {0} hours", e.Hours);
        }
    }

    class Footman
    {
        private string name;

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

        public Footman(string name)
        {
            this.name = name;
        }

        public void Attack(object sender, EventArgs e)
        {
            Console.WriteLine($"Footman {name} is panicking!");
        }
    }
}
```



```
class King
{
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    public King(string name)
    {
        this.name = name;
    }
    public void Attack(object sender, EventArgs e)
    {
        Console.WriteLine($"King {name} is under attack!");
    }

    public event EventHandler AttackedKing;
    public void UnderAttack(EventArgs e)
    {
        AttackedKing(this, e);
    }
}

class RoyalGuard:EventArgs
{
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    public RoyalGuard(string name)
    {
        this.name = name;
    }
    public void Attack(object sender, EventArgs e)
    {
        Console.WriteLine($"Royal Guard {name} is defending!");
    }
}

class Program
{
    static void Main(string[] args)
    {
        King king = new King(Console.ReadLine());
        king.AttackedKing += king.Attack;
        List<RoyalGuard> guards = new List<RoyalGuard>();
        Console.ReadLine().Split().ToList().ForEach(x => guards.Add(new
RoyalGuard(x)));
        List<Footman> footmen = new List<Footman>();
    }
}
```



```
Console.ReadLine().Split().ToList().ForEach(x => footmen.Add(new
Footman(x)));
guards.ForEach(guard => king.AttackedKing += guard.Attack);
footmen.ForEach(fm => king.AttackedKing += fm.Attack);
while (true)
{
    var input = Console.ReadLine().Split().ToArray();
    switch (input[0])
    {
        case "Attack": king.UnderAttack(new EventArgs()); break;
        case "Kill":
            if (guards.Select(x => x.Name).Contains(input[1]))
            {
                var guard = guards.Where(x => x.Name ==
input[1]).First();
                king.AttackedKing -= guard.Attack;
                guards.Remove(guard);
            }
            else
            {
                var fm = footmen.Where(x => x.Name == input[1]).First();
                king.AttackedKing -= fm.Attack;
                footmen.Remove(fm);
            }
            break;
        default: return;
    }
}
}
```

Задача 8.3. Избягване на зависимостите

Даден ви е скелет на прост проект. Проектът съдържа клас примитивен калкулатор, който поддържа два метода - `ChangeStrategy (char @operator)` и `PerformCalculation (int firstOperand, int secondOperand)`. Методът `PerformCalculation` трябва да извършва математически операции върху два операнда, въз основа на текущата стратегия на примитивния калкулатор и `ChangeStrategy` трябва да промени текущата стратегия на калкулатора. В момента Калкулаторът поддържа само добавяне и изваждане на стратегии, помислете как да промените (рефакторирате) `ChangeStrategy` и `PerformCalculation` метод за да позволи на примитивния калкулатор да поддържа всякакви стратегии. Добавете функционалност към примитивния калкулатор, за да поддържа умножение и деление на елементи.

Калкулаторът трябва да стартира по подразбиране в режим `addition` (събиране). Текущият метод `ChangeStrategy` превключва само между 2 стратегии, основани на получаване на символ чрез метод. В момента поддържаните стратегии са:

- "+" за събиране



- “-“ за изваждане

Вход

От клавиатурата вие ще получите редове в един от следните формати, го получаване ан команда “End”:

- “<number> <number>” – изпълнява изчисление на текущите числа с текущия режим изчисление.
- “mode <operator>” – променя режима на изчисляване към указания.

Изход

Извежда резултата от изчислението На всички редове с чила – всеки резултат на нов рег.

Ограничения

- Позволено Ви е да промените (рефактурирате) класа Primitive Calculator, но НЕ Ви е позволено да добавяте допълнителни методи към него като например метод Addition, Subtraction и т.н.
- Операторите, получени от конзолата винаги ще бъдат валидни, указани в секцията за спецификации.
- Резултатът от изчислението също ще бъде цяло число.
- Делител 0 не може да има.
- Винаги се завършва с команда “End”.

Примери

Вход	Изход
10 15 mode / 20 5 17 7 mode - 30 31 End	25 4 2 -1
mode * 1 1 3 21 -5 -6 mode - -30 -50 mode / -28 4 mode + 1 10 End	1 63 30 20 -7 11

Решение

`namespace AvoidDependencies`



```
{
    interface IStrategy
    {
        int Calculate(int first, int second);
    }

    public class Addition : IStrategy
    {
        public int Calculate(int first, int second)
        {
            return first + second;
        }
    }

    public class Division : IStrategy
    {
        public int Calculate(int first, int second)
        {
            return first / second;
        }
    }

    class Multiplying : IStrategy
    {
        public int Calculate(int first, int second)
        {
            return first * second;
        }
    }

    public class Subtraction : IStrategy
    {
        public int Calculate(int firstOperand, int secondOperand)
        {
            return firstOperand - secondOperand;
        }
    }

    public class PrimitiveCalculator
    {
        private IStrategy strategy;

        public PrimitiveCalculator()
        {
            this.strategy = new Addition();
        }

        public void changeStrategy(char @operator)
        {
            switch (@operator)
            {
                case '+':
                    this.strategy = new Addition();
                    break;
                case '-':
```



```
        this.strategy = new Subtraction();
        break;
    case '/':
        this.strategy = new Division();
        break;
    case '*':
        this.strategy = new Multiplying();
        break;
    }
}

public int performCalculation(int firstOperand, int secondOperand)
{
    return strategy.Calculate(firstOperand, secondOperand);
}

}

class Program
{
    static void Main(string[] args)
    {
        PrimitiveCalculator calculator = new PrimitiveCalculator();
        string[] command = Console.ReadLine().Split().ToArray();
        while (command[0] != "End")
        {
            try
            {
                int firstOperand = int.Parse(command[0]);
                int secondOperand = int.Parse(command[1]);
                Console.WriteLine(calculator.performCalculation(firstOperand,
secondOperand));
            }
            catch
            {
                calculator.changeStrategy(command[1].First());
            }
            command = Console.ReadLine().Split().ToArray();
        }
    }
}
```

Задача 8.4. * Работна сила

Създайте два класа - StandartEmployee и PartTimeEmployee и двата да имат име и работни часове седмично. За StandartEmployee работните часове на седмица са винаги 40, а за PartTimeEmployee работните часове на седмица са винаги 20. Създайте клас Job, който трябва да получи служител чрез конструктора си, има полета - name и hours of work required и метод Update, който трябва да изважда от часовете на работа на служителя работните часове на седмица. Когато hours of work required достигне 0 или по-малко трябва да отпечатаме "Job <jobName> done!" и да намерите начин да уведомите колекцията, в която държите всички работи, че е готова и следва да бъдат заличени от колекцията.



Вход

От конзолата ще получите редове в един от следните формати докато получите команда "Край":

- "Job <nameOfJob> <hoursOfWorkRequired> <employeeName>" - ще създаде работа (Job) със указаното име, задължителни седмични и служител.
- "StandartEmployee <name>" – ще създаде стандартен служител с указаното име.
- "PartTimeEmployee <name>" – ще създаде служител с парциално работно време (PartTimeEmployee) с указаното име.
- "Pass Week" – ще извика метода Update на всяка работа(дейност,job).
- "Status" – ще изведе статуса на всички работив сегния формат "Job: <jobName> Hours Remaining: <hoursOfWorkRequired>".

Изход

Винаги всяка работа завършва със съобщението "Job <jobName> done!", което ще се изведе на конзолата. Винаги командата Status показва всички текущо активни работи (незавършени) те ще бъдат изведени в формата, указан за Status, в реда на получаване – всяко съобщение на отделен ред.

Ограничения

- Всички имена съдържат само букви
- Всички задължителни часове на седмица трябва да са положителни цели числа между [1...1000].
- Служителите, указани на входния ред за работа (дейност) винаги трябва да са валидни съществуващи служители.
- Служителите и названията на работите са уникални – няма служители или работи(дейности) с еднакви имена.
- Последната команда винаги е "End".

Примери

Вход	Изход
StandartEmployee Pesho PartTimeEmployee Penka Job FeedTheFishes 45 Pesho Pass Week Status Pass Week End	Job: FeedTheFishes Hours Remaining: 5 Job FeedTheFishes done!
PartTimeEmployee Penka PartTimeEmployee Vanka PartTimeEmployee Stanka Job Something 177 Stanka Pass Week Job AnotherThing 33 Vanka	Job: Something Hours Remaining: 157 Job: AnotherThing Hours Remaining: 33 Job AnotherThing done! Job: Something Hours Remaining: 97



Status Pass Week Pass Week Pass Week Status End	
--	--

Поглед

Намерете начин да имате своя колекция, която да реагира на събития. Създайте свой собствен клас разширяващ ArrayList и реализиращ EventListener за потребителски събития, който се задейства, когато работата е свършена. Използвайте абстракция в класа job за да позволите да бъдат приемани различни типове – например извличете интерфейс за служители и ще имате клас, приемащ като обект от тях, реализиращ интерфейса вместо конкретен клас

Решение

```
namespace TaskForce
{
    abstract class Employee
    {
        string name;
        int workHours;

        public Employee(string n)
        {
            name = n;
        }

        public int WorkHours
        {
            get { return workHours; }
            set { workHours = value; }
        }

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

    class Job
    {
        public Job(Employee employee, string name, int hours)
        {
            baseEmployee = employee;
            this.name = name;
            hoursOfWorkRequired = hours;
        }

        Employee baseEmployee;
    }
}
```



```
string name;
int hoursOfWorkRequired;

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

public void Update()
{
    hoursOfWorkRequired -= baseEmployee.WorkHours;
    if (hoursOfWorkRequired <= 0)
    {
        Console.WriteLine($"Job {name} done!");
    }
}

public override string ToString()
{
    return $"Job: {name} Hours Remaining: {hoursOfWorkRequired}";
}
}

class PartTimeEmployee : Employee
{
    public static int workHours;

    public PartTimeEmployee(string name) : base(name) { base.WorkHours = 20; }
}

class StandartEmployee : Employee
{
    int workHours = 40;

    public StandartEmployee(string name) : base(name) { base.WorkHours = 40; }
}

class Program
{
    static void Main(string[] args)
    {
        string[] input = Console.ReadLine().Split().ToArray();
        List<Employee> employees = new List<Employee>();
        List<Job> jobs = new List<Job>();

        while (input[0] != "End")
        {
            switch (input[0])
            {
                case "StandartEmployee":
                    employees.Add(new StandartEmployee(input[1]));
                    break;
                case "PartTimeEmployee":
                    employees.Add(new PartTimeEmployee(input[1]));
            }
        }
    }
}
```



```
                break;
            case "Job":
                jobs.Add(new Job(employees.FirstOrDefault(n => n.Name ==
input[3]), input[1], int.Parse(input[2])));
                break;
            case "Pass":
                jobs.ForEach(x => x.Update());
                break;
            case "Status":
                jobs.ForEach(x => Console.WriteLine(x.ToString()));
                break;
        }
        input = Console.ReadLine().Split().ToArray();
    }
}
```

Задача 8.5. *Царски гамбит 2

Разширете кода си от проблем 2 King's гамбит - нормално слугите, които сега трябва да умрат в 2 удара (трябва да получи 2 команди Kill с име от входа, които да се убият), докато охранителите трябва да умират от 3 удара. Мъртвите слуги и охранители вече няма да отговарят на събития и трябва да се изтрият от колекцията. Намерете начин за умиращите войниците да съобщят за тяхната смърт на царя и колекцията, която ги съдържа, без ръчно проверка на тяхното състояние на всяка Kill команда (т.е. използвайте събития).

Вход

В първия ред на конзолата ще получите един низ - името на краля. На втория ред ще получите имената на неговите пазачи, разделени с интервали. На третия имената на неговите слуги, разделени с интервали. На следващите редове, докато получите команда "Край", ще получавате команди в един от от следните формати:

- "Attack King" – извиква метод за отговор на атаката срещу царя.
- "Kill <name>" – слугата или охранителя с даденото име, който е атакуван, ако е втора за слуга или трета за охранител – бива убиван

Изход

Когато царят е атакуван трябва да печатате на конзолата "King <kingName> is under attack!" и всеки жив слуга и царски охранител трябва да изведе своето отменно съобщение - първо, всички кралски охранители трябва да отговорят (в реда, в който са въведени) и след това всички слуги трябва да отговорят (в реда, в който са въведени). Всяко съобщение, трябва да бъде отпечатано на нов ред.

Ограничения

- Имената ще съдържат само букви.
- Да има винаги един цар и поне един слуга и един охранител.



- Царят не се убива – няма команда за убиване на царя.
- Командите за убиване се получават само за живи войни
- Всички команди се получават в посочения формат
- Броят на командите ще е положително цяло число между [1...100].
- Последната команда винаги е "End".

Примери

Вход	Изход
Pesho Ruboglav Gosho Stamat Kill Gosho Kill Stamat Attack King Kill Gosho Attack King End	King Pesho is under attack! Royal Guard Ruboglav is defending! Footman Gosho is panicking! Footman Stamat is panicking! King Pesho is under attack! Royal Guard Ruboglav is defending! Footman Stamat is panicking!
HenryVIII Thomas Mark Kill Thomas Kill Mark Attack King Kill Thomas Kill Thomas Kill Mark Attack King End	King HenryVIII is under attack! Royal Guard Thomas is defending! Footman Mark is panicking! King HenryVIII is under attack!

Решение

```
namespace KingGambit2
{
    class Footman
    {
        private string name;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        private int health;

        public int Health
        {
            get { return health; }
            set { health = value; }
        }
        public Footman(string name)
```



```
{
    this.name = name;
    health = 2;
}

public void Attack(object sender, EventArgs e)
{
    if (this.health == 0)
    {
        return;
    }
    health--;
    Console.WriteLine($"Footman {name} is panicking!");
}
public void Death(object sender, EventArgs e)
{
    if (this.health == 0)
    {
        return;
    }
}
}

class King
{
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    public King(string name)
    {
        this.name = name;
    }
    public void Attack(object sender, EventArgs e)
    {
        Console.WriteLine($"King {name} is under attack!");
    }

    public event EventHandler AttackedKing;

    public void UnderAttack(EventArgs e)
    {
        AttackedKing(this, e);
    }
}

class RoyalGuard : EventArgs
{
    private string name;

    public string Name
    {
```



```
        get { return name; }
        set { name = value; }
    }

    private int health;

    public int Health
    {
        get { return health; }
        set { health = value; }
    }

    public RoyalGuard(string name)
    {
        this.name = name;
        health = 3;
    }

    public void Attack(object sender, EventArgs e)
    {
        if (this.health == 0)
        {
            return;
        }
        health--;
        Console.WriteLine($"Royal Guard {name} is defending!");
        if (health == 0) Death(this, e);
    }

    public void Death(object sender, EventArgs e)
    {
        if (this.health == 0)
        {
            return;
        }
        health--;
        Console.WriteLine($"Royal Guard {name} is defending!");
    }
}

class Program
{
    static void Main(string[] args)
    {
        King king = new King(Console.ReadLine());
        king.AttackedKing += king.Attack;

        List<RoyalGuard> guards = new List<RoyalGuard>();
        Console.ReadLine().Split().ToList().
            ForEach(guard => guards.Add(new RoyalGuard(guard)));

        List<Footman> footmen = new List<Footman>();
        Console.ReadLine().Split().ToList().
            ForEach(fm => footmen.Add(new Footman(fm)));
        guards.ForEach(guard => king.AttackedKing += guard.Attack);
        footmen.ForEach(fm => king.AttackedKing += fm.Attack);
        while (true)
```



```
{
    var input = Console.ReadLine().Split().ToArray();
    switch (input[0])
    {
        case "Attack":
            king.UnderAttack(new EventArgs());
            break;
        case "Kill":
            if (guards.Select(x => x.Name).Contains(input[1]))
            {
                var guard = guards.Where(x => x.Name ==
input[1]).First();
                king.AttackedKing -= guard.Attack;
                guards.Remove(guard);
            }
            else
            {
                var fm = footmen.Where(x => x.Name == input[1]).First();
                king.AttackedKing -= fm.Attack;
                footmen.Remove(fm);
            }
            break;
        default:
            return;
    }
}
}
```

Тема 9. Изключения

Задача 9.1. Валидна личност

Дефинирайте прост клас *Person*, притежаващ следните полета: first name, last name и age. Валидирайте данните в setter-ите на свойствата и хвърлете подходящи изключения ако са въведени невалидни данни.

Step 1. Създайте клас *Person*

Създайте проект за това упражнение и добавете клас *Person* в отделен .cs файл. Класът трябва да съдържа следните полета: first name (string), last name (string) and age (int).

Всички полета са задължителни, което означава, че ви трябва един конструктор, който приема и трите като параметри. Например:



```
namespace ExceptionHandling_Exercises
{
    public class Person
    {
        private string firstName;
        private string lastName;
        private int age;

        public Person(string firstName, string lastName, int age)
        {
            // TODO: add properties and validate data
        }

        //TODO: add properties
    }
}
```

Step 2. Добавете свойства и валидирайте данните

Добавете свойство за всяко от полетата. Извършете валидиране в техните setter-и, за да запазите състоянието на обекта Person валидно.

Свойствата first и last name не може да бъдат null или празни низове. За да проверите това, използвайте метода `string.IsNullOrEmpty()`.

Свойството age трябва да е в диапазона [0 ... 120].

Ако са въведени невалидни данни, хвърлете подходящи изключения с достатъчно описателни съобщения. Например ако бъде въведено празно име, едно подходящо изключение би било `ArgumentNullException`. Ако възрастта е отрицателна или твърде голяма, подходящото изключение ще е `ArgumentOutOfRangeException`.

Пример за валидирането на first name (last name е аналогично):



```
public string FirstName
{
    get
    {
        return this.firstName;
    }

    set
    {
        if (string.IsNullOrEmpty(value))
        {
            throw new ArgumentNullException(
                "value",
                "The first name cannot be null or empty.");
        }

        this.firstName = value;
    }
}
```

Пример за валидирането на age:

```
public int Age
{
    get
    {
        return this.age;
    }

    set
    {
        if (value < 0 || 120 < value)
        {
            throw new ArgumentOutOfRangeException(
                "value",
                "Age should be in the range [0 ... 120].");
        }

        this.age = value;
    }
}
```

Сега вече конструкторът би трябвало да използва свойствата вместо да модифицира директно частните полета:



```
public Person(string firstName, string lastName, int age)
{
    this.FirstName = firstName;
    this.LastName = lastName;
    this.Age = age;
}
```

Step 3. Тествайте класът Person

Във вашата главна програма, проверете дали вашият клас се държи както се очаква. Създайте няколко обекта от тип Person – един с валидни данни, един с празно first name, един с null за last name, един с отрицателна възраст и един с age > 120. Проверете дали изпълнението на кода води до грешки, когато се въведат невалидни данни. Тествайте невалидните случаи един по един чрез коментирание на редовете на другите невалидни случаи (защото вашата програма ще спре изпълнението си още при първата грешка).

```
public static void Main()
{
    Person pesho = new Person("Pesho", "Peshev", 24);

    Person noName = new Person(string.Empty, "Goshev", 31);
    Person noLastName = new Person("Ivan", null, 63);
    Person negativeAge = new Person("Stoyan", "Kolev", -1);
    Person tooOldForThisProgram = new Person("Iskren", "Ivanov", 121);
}
```

Step 4. Добавете Try-Catch блокове

За да предотвратите сриването на програмата, обградете невалидните редове в try-catch блокове. Добра практика е да поставите различни catch блокове за различните типове грешки, които очаквате, че командите може да предизвикат. Отпечатайте съобщението на изключението от catch блока.

Пример (невалидно име):



```
try
{
    Person noName = new Person(string.Empty, "Goshev", 31);
}
catch (ArgumentNullException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}
```

```
// Result in console:
// Exception thrown: The first name cannot be null or empty.
// Parameter name: value
```

Пример (невалидна възраст):

```
try
{
    Person negativeAge = new Person("Stoyan", "Kolev", -1);
}
catch (ArgumentNullException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine("Exception thrown: {0}", ex.Message);
}
```

```
// Result in console:
// Exception thrown: Age should be in the range [0 ... 120].
// Parameter name: value
```

Решение

```
namespace Person
{
    public class Person
```



```
{
    private string firstname;
    private string lastname;
    private int age;

    public Person(string FirstName = null, string LastName = null, int Age = -1)
    {
        this.FirstName = FirstName;
        this.LastName = LastName;
        this.Age = Age;
    }

    public int Age
    {
        get { return age; }
        set
        {
            if (value < 0 || value > 120) throw new
ArgumentOutOfRangeException("Age mus be between 0 and 120");
            this.age = value;
        }
    }

    public string LastName
    {
        get { return lastname; }
        set
        {
            if (value == null || value.Length == 0) throw new
ArgumentNullException("Name Cannot be empty");
            this.lastname = value;
        }
    }

    public string FirstName
    {
        get { return firstname; }
        set
        {
            if (value == null || value.Length == 0) throw new
ArgumentNullException("Name Cannot be empty");
            this.firstname = value;
        }
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        try
        {
            Person per = new Person();
        }
        catch (ArgumentException ex)
    }
}
```



```
{
    Console.WriteLine(ex.Message);
}
Console.WriteLine(new string('*', 33));
try
{
    Person per = new Person("", "Stephan", 12);
}
catch (ArgumentException ex)
{
    Console.WriteLine(ex.Message);
}
Console.WriteLine(new string('*', 33));
try
{
    Person per = new Person("Stephi", "Stephanov", -12);
}
catch (ArgumentException ex)
{
    Console.WriteLine(ex.Message);
}
Console.WriteLine(new string('*', 33));

try
{
    Person per = new Person("Stephi", "Stephanov", 12);
    Console.WriteLine("Person Created.");
}
catch (ArgumentException ex)
{
    Console.WriteLine(ex.Message);
}
}
}
```

Задача 9.2. Square Root

Write a program that reads an integer number and calculates and prints its square root. If the number is invalid or negative, print "Invalid number". In all cases finally print "Good bye". Use try-catch-finally.

Решение

```
namespace SquareRoot
{
    public static class CustomMath
    {
        public static double Sqrt(double num)
        {
            if (num < 0) throw new ArgumentException("Value must be possitive");
            double x = num / 2;
            for (int i = 0; i < 100; i++) x = (x + num / x) / 2d;
            return x;
        }
    }
}
```



```
class Program
{
    static void Main(string[] args)
    {
        double result1 = 0;
        double result2 = 0;
        result1 = Math.Sqrt(15);
        try
        {
            result2 = CustomMath.Sqrt(-15);
        }
        catch (ArgumentException ex)
        {
            Console.WriteLine(ex.Message);
        }
        Console.WriteLine($"Original {result1}");
        Console.WriteLine($"Custom {result2}");
    }
}
```

Задача 9.3. Enter Numbers

Write a method ReadNumber(int start, int end) that enters an integer number in given range [start...end]. If an invalid number or non-number text is entered, the method should throw an exception. Using this method write a program that enters 10 numbers: a_1, a_2, \dots, a_{10} , such that $1 < a_1 < \dots < a_{10} < 100$. If the user enters an invalid number, let the user to enter again.

Решение

```
namespace EnterNumbers
{
    class Program
    {
        public static int ReadNumber(int start, int end)
        {
            var num = int.Parse(Console.ReadLine());
            if (num <= start || num > end) throw new ArgumentException($"value must be between {start} - {end}");
            return num;
        }

        static void Main(string[] args)
        {
            List<int> list = new List<int>();
            int n = 0;
            while (list.Count < 10)
            {
                try
                {
                    list.Add(ReadNumber(n, 100));
                    n = list.Last();
                }
                catch (ArgumentException ex)
                {
                }
            }
        }
    }
}
```



```
        Console.WriteLine(ex.Message);  
    }  
    catch (FormatException ex)  
    {  
        Console.WriteLine(ex.Message);  
    }  
    }  
    }  
}
```

Тема 10. Работа с потоци и файлове

Задача 10.1. Нечетни редове

Напишете програма която чете текстов файл и отпечатва на конзолата нечетните редове. Номерацията на редовете започва от 0. Използвайте StreamReader.

text.txt	Изход
-I was quick to judge him, but it wasn't his fault. -Is this some kind of joke?! Is it? -Quick, hide here...It is safer.	-Is this some kind of joke?! Is it?

Решение

```
namespace OddStream  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            StreamReader reader = new StreamReader("file.txt");  
            using (reader)  
            {  
                int lineNumber = 0;  
                string lineContent = reader.ReadLine();  
                while (lineContent != null)  
                {  
                    if (lineNumber % 2 == 1)  
                    {  
                        Console.WriteLine(lineContent);  
                    }  
                    lineNumber++;  
                    lineContent = reader.ReadLine();  
                }  
            }  
        }  
    }  
}
```



Задача 10.2. Номера на редове

Напишете програма, която прочита текстов файл и вмъква номер на ред пред всеки от редовете. Резултатът трябва да бъде записан в друг текстов файл. Използвайте StreamReader в комбинация с StreamWriter.

text.txt	output.txt
-I was quick to judge him, but it wasn't his fault. -Is this some kind of joke?! Is it? -Quick, hide here...It is safer.	Line 1: -I was quick to judge him, but it wasn't his fault. Line 2: -Is this some kind of joke?! Is it? Line 3: -Quick, hide here...It is safer.

Решение

```
namespace LineNumbers
{
    class Program
    {
        static void Main(string[] args)
        {
            using (StreamReader reader = new StreamReader("file.txt"))
            {
                using (StreamWriter writer = new StreamWriter("output.txt"))
                {
                    int lineNumber = 1;
                    string lineContent = reader.ReadLine();
                    while (lineContent != null)
                    {
                        writer.WriteLine($"Line {lineNumber}: {lineContent}");
                        lineNumber++;
                        lineContent = reader.ReadLine();
                    }
                }
            }
            using (StreamReader reader = new StreamReader("output.txt"))
            {
                Console.WriteLine(reader.ReadToEnd());
            }
        }
    }
}
```

Задача 10.3. Брой на думите

Напишете програма, която прочита списък от думи от файла words.txt и намира колко пъти всяка от думите е използвана в друг файл text.txt. Сравняването трябва да бъде нечувствително към регистъра (малки/главни букви).

Запишете резултатите във файл results.txt. Сортирайте думите според честотата на използването им, в намаляващ ред. Използвайте StreamReader в комбинация със StreamWriter.



words.txt	text.txt	result.txt
quick is fault	-I was quick to judge him, but it wasn't his fault. -Is this some kind of joke?! Is it? -Quick, hide here...It is safer.	is - 3 quick - 2 fault - 1

Решение

```
namespace WordsCounter
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<string, int> dict = new Dictionary<string, int>();
            using (StreamReader words = new StreamReader("words.txt"))
            {
                using (StreamReader text = new StreamReader("text.txt"))
                {
                    using (StreamWriter writer = new StreamWriter("output.txt"))
                    {
                        var line = text.ReadToEnd().ToLower().Split(' ', '?', ',', '.', '-', '!').ToArray();
                        var search = words.ReadLine();

                        while (search != null)
                        {
                            dict.Add(search, line.Count(x => x == search));
                            search = words.ReadLine();
                        }
                        dict = dict.OrderByDescending(x => x.Value).ToDictionary(x => x.Key, x => x.Value);

                        foreach (var item in dict)
                        {
                            writer.WriteLine($"{item.Key} - {item.Value}");
                        }
                    }
                }
            }
            using (StreamReader reader = new StreamReader("output.txt"))
            {
                Console.WriteLine(reader.ReadToEnd());
            }
        }
    }
}
```

Задача 10.4. Копиране на двоичен файл

Напишете програма, която копира съдържанието на двоичен файл (например изображение, видео и т.н.) в друг с помощта на FileStream. Не ви е позволено да използвате класа File или други подобни помощни класове.









Решение

```
namespace BinaryFileCopy
{
    class Program
    {
        static void Main(string[] args)
        {
            using (FileStream source = new FileStream("file.png", FileMode.Open))
            {
                using (FileStream destination = new FileStream("newfile.png",
FileMode.Create))
                {
                    byte[] bytes = new byte[4096];
                    while (true)
                    {
                        int data = source.Read(bytes, 0, bytes.Length);
                        if (data == 0) break;
                        destination.Write(bytes, 0, bytes.Count());
                    }
                }
            }
        }
    }
}
```

Задача 10.5. Разделяне на файл на части

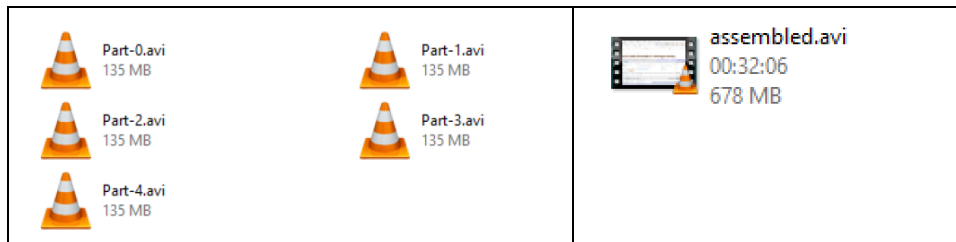
Напишете програма, която взема произволен файл и го нарязва на n части. Разпишете следните методи:

- Slice(string sourceFile, string destinationDirectory, int parts) - разделя дадения sourceFile на n части и ги записва в destinationDirectory.

Входен файл	Изходяща директория
<p>parts = 5</p>  <p>SOLID-Logger.avi 00:32:06 678 MB</p>	<div>  Part-0.avi 135 MB </div> <div>  Part-2.avi 135 MB </div> <div>  Part-4.avi 135 MB </div> <div>  Part-1.avi 135 MB </div> <div>  Part-3.avi 135 MB </div>

- Assemble(List<string> files, string destinationDirectory) - комбинира всички файлове в един, в реда, в който са подадени, и записва резултата в destinationDirectory.

Входни файлове	Изходяща директория
----------------	---------------------



Използвайте `FileStream`. Не ви е позволено да използвате класа `Файл` или други подобни помощни класове.

Решение

```
namespace FilePartition
{
    internal class Program
    {
        private static List<string> Slice(string sourceFile, int parts)
        {
            var files = new List<string>();
            var reversed = new string(sourceFile.Reverse().ToArray());
            string ext = new string(reversed.Substring(0,
reversed.IndexOf(".").Reverse().ToArray()));
            using (FileStream read = new FileStream(sourceFile, FileMode.Open))
            {
                int FileLength = (int)read.Length;
                int sizeOfFile = FileLength / parts;
                for (int i = 0; i < parts; i++)
                {
                    var fileName = $"partition_{i}.{ext}";
                    files.Add(fileName);
                    using (FileStream write = new FileStream($"{{fileName}}",
FileMode.Create))
                    {
                        byte[] bytes = new byte[sizeOfFile];
                        int data = read.Read(bytes, 0, bytes.Length);
                        write.Write(bytes, 0, bytes.Count());
                    }
                }
            }
            return files;
        }

        public static void Assemble(List<string> files)
        {
            string ext = files.First().Substring(files.First().IndexOf(".") + 1);
            using (FileStream write = new FileStream($"assembled.{ext}",
FileMode.Create))
            {
                foreach (var item in files)
                {
                    using (FileStream read = new
FileStream(string.Format($"{{item}}"), FileMode.Open))
                    {
```



```

        byte[] bytes = new byte[4096];
        while (true)
        {
            int bytesRead = read.Read(bytes, 0, bytes.Length);
            if (bytesRead == 0) break;
            write.Write(bytes, 0, bytesRead);
        }
    }
}








static void Main(string[] args)
{
    var files = Slice("video.mp4", 4);
    Console.WriteLine("Step 1. Slice Completed.");
    Assemble(files);
    Console.WriteLine("Step 2. Assemble Completed.");
}
}
}

```

Задача 10.6. Компресиране на нарязаните файлове

Променете вашата предходна програма, така че тя също така да компресира байтовете докато ги разделя на части и да ги разкомпресира когато ги обединява пак в оригиналния файл. Използвайте GzipStream.

Съвет: Когато взимате файловете от папката, подсигурете се, че взимате само тези с .gz разширение (защото може например да има и скрити файлове).

Source File	Compressed & Sliced	Decompressed & Assembled
<p>parts = 5</p>  <p>SOLID-Logger.avi 00:32:06 678 MB</p>	<div>  Part-0.gz GZ File 115 MB </div> <div>  Part-2.gz GZ File 113 MB </div> <div>  Part-4.gz GZ File 111 MB </div> <div>  Part-1.gz GZ File 112 MB </div> <div>  Part-3.gz GZ File 114 MB </div>	 <p>assembled.avi 00:32:06 678 MB</p>

Решение

`namespace Compression`

```

{
    class Program
    {
        private static void Slice(string sourceFile, int parts)
        {
            using (FileStream read = new FileStream(sourceFile, FileMode.Open))
            {
                int sizeOfFile = ((int)read.Length) / parts;
            }
        }
    }
}

```



```
        for (int i = 0; i < parts; i++)
        {
            using (FileStream write = new FileStream($"part_{i}",
FileMode.Create))
            {
                using (GZipStream gz = new GZipStream(write,
CompressionMode.Compress, false))
                {
                    byte[] bytes = new byte[sizeOfFile];
                    int data = read.Read(bytes, 0, bytes.Length);
                    gz.Write(bytes, 0, bytes.Length);
                }
            }
        }
    }

    public static void Assemble()
    {
        string[] files =
Directory.GetFiles(Directory.GetCurrentDirectory().ToString(), "part*");
        using (FileStream write = new FileStream($"assembled", FileMode.Create))
        {
            foreach (var item in files)
            {
                using (FileStream read = new FileStream(item, FileMode.Open))
                {
                    byte[] bytes = new byte[4096];
                    using (GZipStream decompression = new GZipStream(read,
CompressionMode.Decompress))
                    {
                        decompression.CopyTo(write);
                    }
                }
            }
        }
    }

    static void Main(string[] args)
    {
        int parts = 4;
        Slice("video.mp4", parts);
        Console.WriteLine("Step 1. Slice Completed.");
        Assemble();
        Console.WriteLine("Step 2. Assemble Completed.");
    }
}
```

Задача 10.7. Претърсване на директория

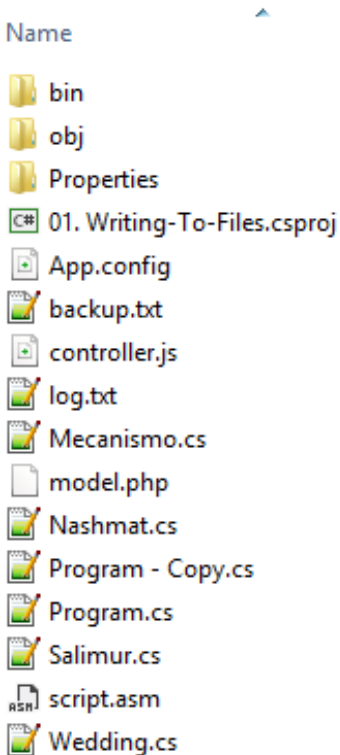
Претърсете дадена директория за всички файлове с дадено разширение. Търсете само в първото ниво на папката и опишете информацията за всеки намерен файл във report.txt.



Файловете трябва да бъдат групирани според своето разширение. Разширенията трябва да са подредени според броя на файловете, които притежават такова разширение, в намаляващ ред, и по името в азбучен ред.

Файловете с едно и също разширение трябва да бъдат подредени според своя размер.

report.txt трябва да бъде записан на работния плот. Подсигурете се, че винаги използвате валидния път към тази папка, независимо кой е логнатия потребител.

Вход	Изглед към папката	report.txt
.		<pre>.cs --Mecanismo.cs - 0.994kb --Program.cs - 1.108kb --Nashmat.cs - 3.967kb --Wedding.cs - 23.787kb --Program - Copy.cs - 35.679kb --Salimur.cs - 588.657kb .txt --backup.txt - 0.028kb --log.txt - 6.72kb .asm --script.asm - 0.028kb .config --App.config - 0.187kb .csproj --01. Writing-To-Files.csproj - 2.57kb .js --controller.js - 1635.143kb .php --model.php - 0kb</pre>

Решение

```
namespace FolderSearch
{
    class Program
    {
        static void Main(string[] args)
        {
            string searched = "Program.cs";
            Console.WriteLine("Searching for {0}", searched);

            DirectoryInfo d = new DirectoryInfo(Directory.GetCurrentDirectory());
            FileInfo[] fileInfo = d.GetFiles($"*{searched}*");
```



```
fileInfo = fileInfo.OrderBy(x => x.Extension).ThenBy(x =>
x.Length).ToArray();

using (StreamWriter writer = new StreamWriter("report.txt"))
{
    string lastExt = string.Empty;
    foreach (FileInfo file in fileInfo)
    {
        if (lastExt != file.Extension)
        {
            lastExt = file.Extension;
            writer.WriteLine($"{lastExt}");
        }
        writer.WriteLine($"{file.Name}.{file.Extension} - {file.Length
/ 1000}kb");
    }
}

using (var reader = new StreamReader("report.txt"))
{
    Console.WriteLine("report.txt\n---\n{0}", reader.ReadToEnd());
}
}
```

Задача 10.8. * Пълно претърсване на директория

Променете предходната си програма, така че тя да обхожда рекурсивно и всички поддиректории на първоначалната директория.

Решение

```
namespace FullFolderSearch
```

```
{
    class Program
    {
        public static string searched = "Program.cs";
        public static string location = "C:\\Users\\mitko\\Desktop";

        static void GetInfo(DirectoryInfo dir)
        {
            DirectoryInfo[] ad = dir.GetDirectories();
            if (ad.Count() != 0)
            {
                foreach (var item in ad)
                {
                    GetInfo(item);
                }
            }
            FileInfo[] fileInfo = dir.GetFiles($"*{searched}*");
            fileInfo = fileInfo.OrderBy(x => x.Extension).ThenBy(x =>
x.Length).ToArray();

            using (StreamWriter writer = new StreamWriter("report.txt", true))
            {
```



```
string lastExt = string.Empty;
foreach (FileInfo file in fileInfo)
{
    if (lastExt != file.Extension)
    {
        lastExt = file.Extension;
        writer.WriteLine($".{lastExt}");
    }
    writer.WriteLine($"---{file.Name}.{file.Extension} - {file.Length
/ 1000}kb");
}
}

static void Main(string[] args)
{
    Console.WriteLine($"Searching for {searched} in {location} ...");

    DirectoryInfo dir = new DirectoryInfo(location);
    GetInfo(dir);
    DirectoryInfo[] ad = dir.GetDirectories();
    foreach (var item in ad)
    {
        GetInfo(item);
    }

    using (var reader = new StreamReader("report.txt"))
    {
        Console.WriteLine("report.txt\n---\n{0}", reader.ReadToEnd());
    }
}
}
```

Задача 10.9. ** HTTP сървър

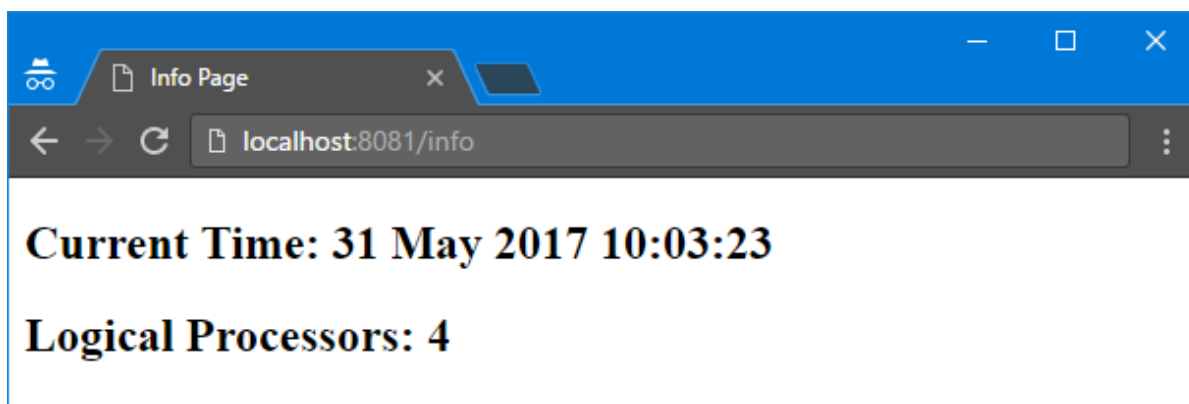
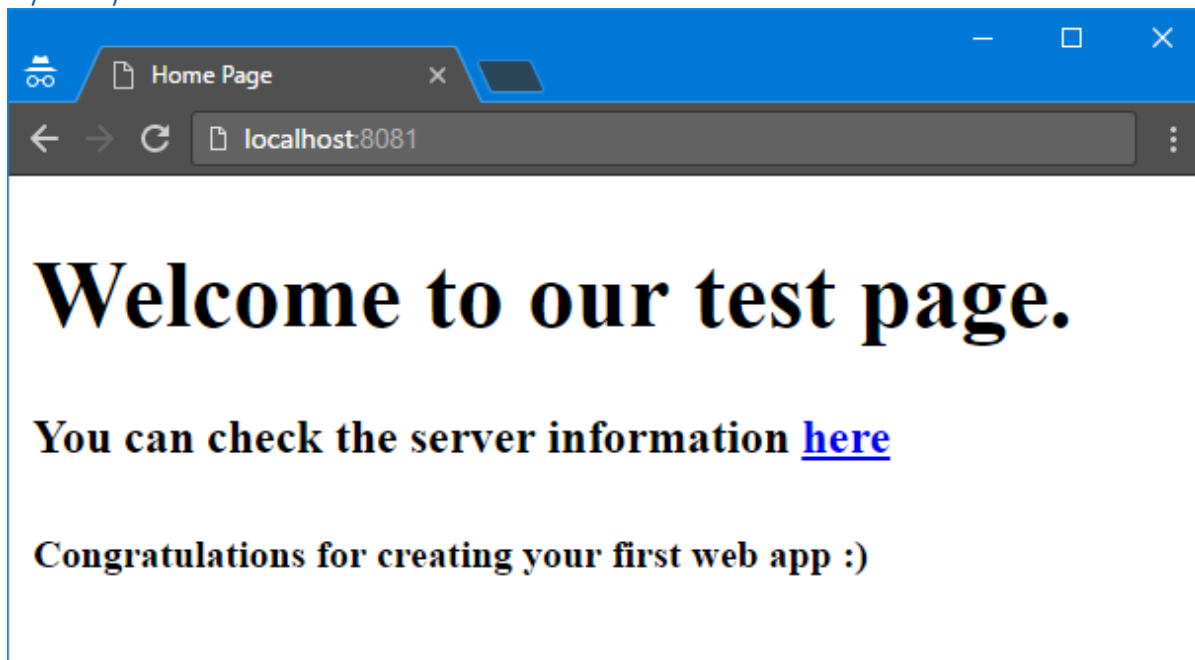
Създайте прост HTTP сървър, който ще може да получава заявки и да връща подходящи отговори според заявения път. Прочетете малко повече в Интернет как трябва да изглеждат HTTP заявка и нейния отговор. Създайте сайт с 3 страници:

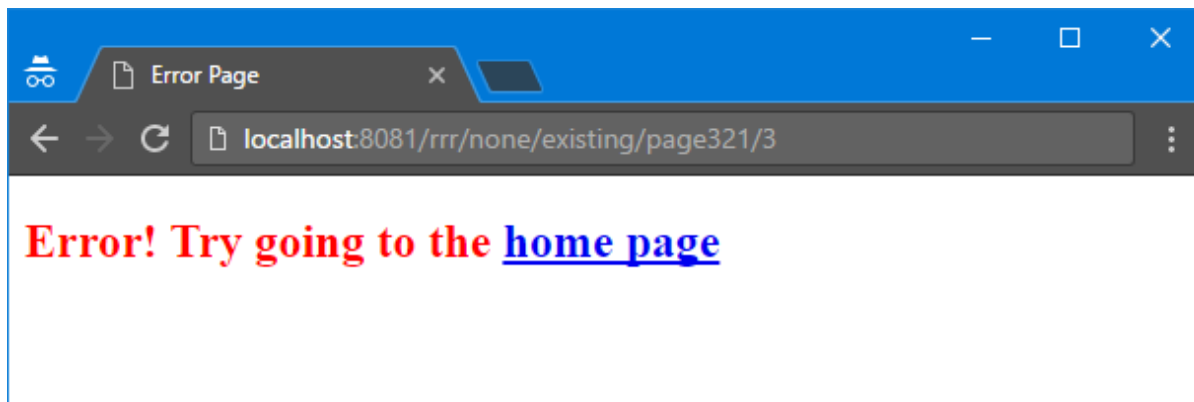
- 1^{вата} страница трябва да е достъпна на адрес **localhost:{port}/** - (това е основната директория). Тази страница съдържа само приветствено съобщение и препратка към втората страница
- 2^{рата} страница трябва да е достъпна на адрес **localhost:{port}/info** – тази страница показва текущото време и броят на логическите процесори в компютъра
- 3^{тата} трябва да бъде страница за грешки - ако потребителят се опита да достъпи каква да е друга страница върнете като отговор тази страница за грешки.



Ще ви бъдат предоставени за улеснение HTML файлове на страниците, които ни трябва. Можете да ги променяте както прецените или дори да създадете ваши собствени HTML файлове.

Примери





Решение

```
namespace HttpServer
{
    class Program
    {
        static void Main(string[] args)
        {
            // Step 1. Create TCP network connection listener
            Console.WriteLine("http://127.0.0.1:8080/");
            TcpListener listener = new TcpListener(IPAddress.Any, 8080);
            listener.Start();

            while (true)
            {
                // Step 2. Accept TCP network client
                using (NetworkStream stream =
                    listener.AcceptTcpClient().GetStream())
                {
                    // Step 3. Read 4K bytes
                    byte[] requestBytes = new byte[4096];
                    stream.Read(requestBytes, 0, 4096);
                    Console.WriteLine(Encoding.UTF8.GetString(requestBytes));

                    // Step 4. Find requested page
                    var requestString = Encoding.ASCII.GetString(requestBytes);
                    string page = string.Empty;
                    try
                    {
                        page = requestString.Substring(requestString.IndexOf("/") +
1, requestString.IndexOf("HTTP/") - requestString.IndexOf("/") - 1).Trim();
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine($"Error: {e.Message}");
                    }

                    // Step 5. Read html page
                    StreamReader reader = null;
                    string code = string.Empty;
                    switch (page)
```



```
{
    case "page.html":
    case "error.html":
    {
        reader = new StreamReader(page);
        code = reader.ReadToEnd();
        break;
    }

    case "info.html":
    {
        reader = new StreamReader(page);
        code = reader.ReadToEnd();
        code = code.Replace("{Date}",
${DateTime.Now.Day}.{DateTime.Now.Month}.{DateTime.Now.Year},
{DateTime.Now.Hour}:{DateTime.Now.Minute}:{DateTime.Now.Second}");
        code = code.Replace("{CPU}",
Environment.ProcessorCount.ToString());
        break;
    }
    default:
    {
        reader = new StreamReader("page.html");
        code = reader.ReadToEnd();
        break;
    }
}

// Step 6. Form the response
StringBuilder message = new StringBuilder();
message.Append("HTTP/1.1 200 OK\r\n");
message.Append("Content-Type: text/html\r\n");
message.Append("Content-Length: " + code.Length + "\r\n\r\n");
message.Append(code);

// Step 7. Return Response to Client
byte[] bytes = Encoding.ASCII.GetBytes(message.ToString());
stream.Write(bytes, 0, bytes.Length);

// Step 8. Clean Up
stream.Flush();
stream.Close();
}
}
}
}
```



Съдържание

Модул 5. Обектно-ориентирано програмиране.....	1
Тема 1. Компонентно тестване.....	1
Задача 1.1. Скелет	1
Задача 1.2. NUnit тестове на Ахе.....	3
Задача 1.3. MSTest тестове на Ахе.....	4
Задача 1.4. NUnit тестове на Dummy.....	5
Задача 1.5. MSTest тестове на Dummy	6
Задача 1.6. Фалшиви Ахе и Dummy.....	8
Задача 1.7. Имитиране.....	11
Тема 2. Дефиниране на класове.....	11
Задача 2.1. Дефиниране на клас човек	11
Задача 2.2. Семейство.....	12
Задача 2.3. Статистика.....	14
Задача 2.4. Дефиниране на клас Рационално число.....	16
Задача 2.5. Несъкратима дроб *	17
Задача 2.6. Дефиниране на клас Четно число *	19
Задача 2.7. Дефиниране на клас Четно число **	19
Задача 2.8. Дефиниране на клас Нечетно число.....	20
Задача 2.9. Дефиниране на клас Кратно на „к“ число	20
Тема 3. Шаблонни класове.....	21
Задача 3.1. Кутия с Т	21
Задача 3.2. Кутия за всичко	22
Задача 3.3. Универсална кутия за низове	24
Задача 3.4. Универсална кутия за цели числа.....	25
Задача 3.5. Създател на шаблонен масив	27
Задача 3.6. Шаблонен метод за размяна на низове.....	28
Задача 3.7. Шаблонен метод за размяна на цели числа	29
Задача 3.8. Шаблонен метод за броене на низове.....	30
Задача 3.9. Шаблонен метод за броене на дробни числа.....	32
Задача 3.10. Универсална везна	34
Задача 3.11. Подобрен списък.....	35
Задача 3.12. Сортиране на подобрения списък.....	38



Задача 3.13. *Обхождане на подобрения списък	41
Задача 3.14. Tuple	44
Задача 3.15. Threuple	46
Тема 4. Наследяване, абстракция, интерфейси	49
Задача 4.1. Единично наследяване	49
Задача 4.2. Наследяване на много нива	49
Задача 4.3. Йерархично наследяване	50
Задача 4.4. Случаен списък	52
Задача 4.5. Стек от низове	52
Задача 4.6. Фигури	54
Задача 4.7. Коли	57
Задача 4.8. Дефиниране на интерфейс IPerson	59
Задача 4.9. Множествена имплементация	60
Задача 4.10. Телефония	62
Задача 4.11. Работници	64
Задача 4.12. Животинско царство	68
Задача 4.13. Граничен контрол	70
Задача 4.14. Рожден ден	72
Задача 4.15. Недостиг на храна	75
Тема 5. Полиморфизъм	79
Задача 5.1. Фигури	79
Задача 5.2. Животинско царство 2	81
Задача 5.3. Превозни средства	83
• На първи ред – информация за колата във формат {Car {fuel quantity} {liters per km}}	84
• На втори ред – информация за камиона във формат {Truck {fuel quantity} {liters per km}}	84
• На трети ред – брой команди N, които ще бъдат подадени на следващите N реда	84
• На следващите N реда – команди във формат:	84
○ Drive Car {distance}	84
○ Drive Truck {distance}	84
○ Refuel Car {liters}	84
○ Refuel Truck {liters}	84



След всяка Drive команда отпечатайте дали колата/камионът може да пропътува разстоянието, като използвате следния формат при успех:.....	84
Задача 5.4. Превозни средства 2	88
• На четвъртия рег – броят на командите N, които ще бъдат подадени на следващите N рега	88
• На следващите N рега – команди във формата.....	88
○ Drive Car {distance}	88
○ Drive Truck {distance}.....	88
○ Drive Bus {distance}	88
○ DriveEmpty Bus {distance}.....	88
○ Refuel Car {liters}.....	89
○ Refuel Truck {liters}.....	89
○ Refuel Bus {liters}.....	89
След всяка Drive команда отпечатайте дали колата/камионът/автобусът може да пропътува това разстояние във формат при успех:.....	89
Или при неуспех:.....	89
Ако даденото гориво е	89
Ако даденото гориво, не може да се вмести в резервоара, отпечатайте "Cannot fit in tank"	89
Накрая, отпечатайте оставащото гориво за колата, камиона и автобуса, закръглени до 2 знака след запетаята във формат:.....	89
Задача 5.5. Ферма за животни	95
Тема 6. Работа с обекти	101
Задача 6.1. Библиотека	101
Задача 6.2. Итератор за библиотека	102
Задача 6.3. ListIterator	104
Задача 6.4. Колекция.....	107
Задача 6.5. Стек.....	108
Задача 6.6. Жабче	112
Задача 6.7. *Клиника за домашни любимци	114
Задача 6.8. ***Обхождане на свързан списък.....	124
Задача 6.9. Сравнима книга	128
Задача 6.10. Сравняващият книги	130



Задача 6.11. Сравняване на обекти	133
Задача 6.12. Шаблон Strategy	135
Задача 6.13. *Логика за еднаквост.....	138
Задача 6.14. Крадец.....	141
Задача 6.15. Висококачествени грешки.....	143
Задача 6.16. Мисия „Частно“ невъзможна.....	145
Задача 6.17. Колектор.....	147
Задача 6.18. Поля за жътва	149
Задача 6.19. Black Box Integer	153
Задача 6.20. BarracksWars – Нова фабрика	156
Задача 6.21. BarracksWars – Командите отвърщат на удара.....	163
Задача 6.22. * BarracksWars - Завръщане на Зависимостите.....	165
Тема 7. Елементи от функционалното програмиране.....	166
Задача 7.1. Сортиране на нечетни числа	166
Задача 7.2. Сбор на числа	166
Задача 7.3. Брой думи главна буква.....	167
Задача 7.4. Начисляване на ДДС (VAT).....	168
Задача 7.5. Филтриране по възраст	168
Задача 7.6. Action	170
Задача 7.7. Рицари на честта.....	171
Задача 7.8. Собствена Min функция	171
Задача 7.9. Намерете четно или нечетно	172
Задача 7.10. Приложна аритметика	173
Задача 7.11. Обърни и изпълни	174
Задача 7.12. Предикати за имена.....	174
Задача 7.13. Потребителски сравнител.....	175
Задача 7.14. Списък от предикати.....	176
Задача 7.15. Предикатно парти!.....	176
Задача 7.16. Модул Филтър за резервация на парти.....	178
Задача 7.17. Inferno III.	179
Задача 7.18. TriFunction.....	181
Тема 8. Комуникация между обекти	182
Задача 8.1. Реализация на събитие	182



Задача 8.2. Царски гамбит.....	184
Задача 8.3. Избягване на зависимостите	188
Задача 8.4. * Работна сила	191
Задача 8.5. *Царски гамбит 2.....	195
Тема 9. Изключения.....	199
Задача 9.1. Валидна личност.....	199
Задача 9.2. Square Root.....	205
Задача 9.3. Enter Numbers	206
Тема 10. Работа с потоци и файлове.....	207
Задача 10.1. Нечетни редове.....	207
Задача 10.2. Номера на редове	208
Задача 10.3. Брой на думите	208
Задача 10.4. Копиране на двоичен файл	209
Задача 10.5. Разделяне на файл на части.....	210
Задача 10.6. Компресиране на нарязаните файлове	212
Задача 10.7. Претърсване на директория.....	213
Задача 10.8. * Пълно претърсване на директория.....	215
Задача 10.9. ** HTTP сървър.....	216