Модул 3. Увод в ООП

Тема 1. Дефиниране на класове

Задача 1.1. Дефиниране на клас Person

Създайте клас Person, който трябва да има:

- 2 полета: name: String и age: int - поле
- 2 свойства: Name: String и Age: int – свойство
- 1 метод: IntroduceYourself

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/228#0

Решение

```
namespace PersonClass
{
    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("My name is {0} and I am {1} years old.", name, age);
        }
    }

    class Program
    {
        static void Main()
        {
            Person firstPerson = new Person();
            firstPerson.Name = "Ivan";
            firstPerson.Age = 12;
            firstPerson.IntroduceYourself();
        }
    }
}
```

Задача 1.2. Дефиниране на клас BankAccount (банкова сметка)

Създайте клас **BankAccount**, който трябва да има:

- 2 полета: id: int и balance: double
- `2 свойства: Id: int и` Balance: double

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/674#0

Решение

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

        public int Id
        {
            get { return id; }
            set { id = value; }
        }

        private double balance;

        public double Balance
        {
            get { return balance; }
            set { balance = value; }
        }
    }

    class Program
    {
        static void Main()
        {
           BankAccount account = new BankAccount();
           account.Id= 1;
           account.Balance = 15;
           Console.WriteLine($"Account {account.Id}, balance {account.Balance}");
        }
    }
}
```

Задача 1.3. Методи

Надградете класа **BankAccount.** Създайте методите:

- `Deposit(Double amount): void – който да вкарва пари в сметката`
- `Withdraw(Double amount): void – който да изтегля пари от сметката`
- Заменете метода ToString()

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/674#1

Решение

```
namespace BankAccountMethods
{
```

```
    class BankAccount
    {
        private int id;

        public int Id
        {
            get { return id; }
            set { id = value; }
        }

        private double balance;

        public double Balance
        {
            get { return balance; }
            set { balance = value; }
        }

        public void Deposit(double amount)
        {
            this.balance += amount;
        }

        public void Withdraw(double amount)
        {
            this.balance -= amount;
        }

        public override string ToString()
        {
            return $"Account {this.id}, balance {this.balance}";
        }
    }

    class Program
    {
        static void Main()
        {
           BankAccount account = new BankAccount();
           account.Deposit(15);
           account.Withdraw(5);
           Console.WriteLine(account.ToString());
        }
    }
}
```

Задача 1.4. Тестов Клиент

Създайте тестов клиент, който използва **BankAccount**.

Трябва да поддържате следните операции:

- Create {Id}
- Deposit {Id} {Amount}
- Withdraw {Id} {Amount}

- Print {Id}
- End

Ако се опитате да създадете сметка със съществуващо Id, изведете "Account already exists".

Ако се опитате да извършите операция върху несъществуваща сметка, изведете "Account does not exist".

Ако се опитате да изтеглите сума, която е по-голяма от баланса, изведете "Insufficient balance".

Print командата, трябва да изведе "Account ID{id}, balance {balance}". Закръглете баланса до втория знак след запетаята.

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/674#2

Примери

Вход	Изход
Create 1 Create 2 Deposit 1 20 Withdraw 1 30 Withdraw 1 10 Print 1 End	Account already exists Insufficient balance Account ID1, balance 10.00
Create 1 Deposit 2 20 Withdraw 2 30 Print 2 End	Account does not exist Account does not exist Account does not exist

Решение

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

        public int Id
        {
            get { return id; }
            set { id = value; }
        }

        private double balance;

        public double Balance
        {
            get { return balance; }
            set { balance = value; }
```

```
        }

        public void Deposit(double amount)
        {
            this.balance += amount;
        }

        public void Withdraw(double amount)
        {
            this.balance -= amount;
        }

        public override string ToString()
        {
            return $"Account ID{this.id}, balance {this.balance:f2}";
        }
    }

    class Client
    {
        Dictionary<int, BankAccount> accounts = new Dictionary<int, BankAccount>();

        public void Create(int id)
        {
            if (accounts.ContainsKey(id))
            {
                Console.WriteLine("Account already exists");
            }
            else
            {
                var account = new BankAccount();
                account.Id = id;
                accounts.Add(id, account);
            }
        }

        public void Deposit(int id, double balance)
        {
            if (!accounts.ContainsKey(id))
            {
                Console.WriteLine("Account does not exist");
            }
            else
            {
                var account = accounts.First(x => x.Key == id).Value;
                account.Deposit(balance);
            }
        }

        public void Withdraw(int id, double balance)
        {
            if (!accounts.ContainsKey(id))
            {
                Console.WriteLine("Account does not exist");
            }
```

```
            else
            {
                var account = accounts.First(x => x.Key == id).Value;
                if (account.Balance < balance)
                {
                    Console.WriteLine("Insufficient balance");
                }
                else
                {
                    account.Withdraw(balance);
                }
            }
        }

        public void Print(int id)
        {
            if (!accounts.ContainsKey(id))
            {
                Console.WriteLine("Account does not exist");
            }
            else
            {
                var account = accounts.First(x => x.Key == id).Value;
                Console.WriteLine(account.ToString());
            }
        }
    }

    class Program
    {
        static void Main()
        {
            Client client = new Client();

            var cmd = Console.ReadLine().Split().ToArray();

            while (cmd[0] != "End")
            {
                switch (cmd[0])
                {
                    case "Create":
                        client.Create(int.Parse(cmd[1]));
                        break;

                    case "Deposit":
                        client.Deposit(int.Parse(cmd[1]), double.Parse(cmd[2]));
                        break;

                    case "Withdraw":
                        client.Withdraw(int.Parse(cmd[1]), double.Parse(cmd[2]));
                        break;

                    case "Print":
                        client.Print(int.Parse(cmd[1]));
                        break;
```

```
                }
                cmd = Console.ReadLine().Split().ToArray();
            }
        }
    }
}
```

Задача 1.5. Човекът и неговите пари

Създайте клас Person. Той трябва да има полета за:

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

Класът трябва да има метод, който изчислява всички пари, които притежава човека от сметките си:

- **`GetBalance(): double`**

Създайте метод **GetBalance()**, който намира сумата по всички сметки.

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/674#3

Решение

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

        public int Id
        {
            get { return id; }
            set { id = value; }
        }

        private double balance;

        public double Balance
        {
            get { return balance; }
            set { balance = value; }
        }
    }

    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; }
        }

        private List<BankAccount> accounts = new List<BankAccount>();

        public List<BankAccount> Accounts
        {
            get { return accounts; }
            set { accounts = value; }
        }

        public double GetBalance()
        {
            return accounts.Sum(element => element.Balance);
        }
    }

    class Program
    {
        static void Main()
        {
            Person Peter = new Person();
            Peter.Name = "Peter Petrov";
            Peter.Age = 45;

            Peter.Accounts = new List<BankAccount>()
            {
                new BankAccount()
                {
                    Id = 321,
                    Balance = 543.21
                },
                new BankAccount()
                {
                    Id = 413,
                    Balance = 1322.32
                }
            };

            Console.WriteLine("Person: {0}, Total Balance = {1}", Peter.Name,
Peter.GetBalance());
        }
    }
}
```

Задача 1.6. Дефиниране на клас човек

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

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/228#0

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.

Решение

```
using System.Reflection;

namespace PersonReflection
{
    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; }
        }
    }

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

Бонус

Опитайте се да създадете няколко обекта от Person:

Име	Възраст
Pesho	20
Gosho	18
Stamat	43

Задача 1.7. 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.

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/228#2

Примери

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

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

*Бонус**

Опитайте се да създадете метод Print за класа Family

Решение

```
namespace PersonFamily
{
    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; }
        }
    }

    class Family
    {
        private List<Person> members = new List<Person>();
```

```
        public void AddMember(Person member)
        {
            members.Add(member);
        }

        public void Print()
        {
            foreach (Person member in members)
            {
                Console.WriteLine($"{member.Name} - {member.Age}");
            }
        }
    }

    class Program
    {
        static void Main()
        {
            Family family = new Family();

            int n = int.Parse(Console.ReadLine());
            while (n > 0)
            {
                var line = Console.ReadLine().Split();
                family.AddMember(new Person()
                {
                    Name = line[0],
                    Age = int.Parse(line[1])
                });
                n--;
            }

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

Задача 1.8. 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.

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/228#2

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
{
    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; }
        }
    }

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

        public void AddMember(Person member)
        {
            members.Add(member);
        }

        public void Print()
        {
            members = members.Where(x => x.Age > 30).OrderBy(y => y.Name).ToList();

            foreach (Person member in members)
            {
                Console.WriteLine($"{member.Name} - {member.Age}");
            }
        }
    }

    class Program
    {
        static void Main()
        {
            Family family = new Family();
```

```
            int n = int.Parse(Console.ReadLine());
            while (n > 0)
            {
                var line = Console.ReadLine().Split();
                family.AddMember(new Person()
                {
                    Name = line[0],
                    Age = int.Parse(line[1])
                });
                n--;
            }

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

Тема 2. Полета и методи

Задача 2.1. Списък на служители

Дефинирайте клас Employee, съдържащ информация за име, заплата, длъжност, отдел, електронна поща и възраст. Полетата име, заплата, длъжност и отдел са задължителни, останалите са опционални.

Вашата задача е да напишете програма, която прочита N реда с информация за служители от конзолата, намира кой е отдела с най-висока средна заплата и за всеки служител от този отдел отпечатва неговото име, заплата, електронна поща и възраст. Служителите трябва да са сортирани според заплатите им, в намаляващ ред. Ако някой служител няма електронна поща, на нейно място трябва да се отпечата "n/a", а ако няма указана възраст, да се изведе "-1" вместо това. Заплатата трябва да бъде отпечатана с две цифри след десетичния знак.

Примери

Вход	Изход
4 Pesho 120.00 Dev Development pesho@abv.bg 28 Toncho 333.33 Manager Marketing 33 Ivan 840.20 ProjectLeader Development ivan@ivan.com Gosho 0.20 Freeloader Nowhere 18	Highest Average Salary: Development Ivan 840.20 ivan@ivan.com -1 Pesho 120.00 pesho@abv.bg 28
6 Stanimir 496.37 Temp Coding stancho@yahoo.com Yovcho 610.13 Manager Sales Toshko 609.99 Manager Sales toshko@abv.bg 44	Highest Average Salary: Sales Yovcho 610.13 n/a -1 Toshko 609.99 toshko@abv.bg 44

Venci 0.02 Director BeerDrinking beer@beer.br 23 Andrei 700.00 Director Coding Popeye 13.3333 Sailor SpinachGroup popeye@pop.ey	

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/228#5

Решение

```
namespace Employees
{
    class Employee
    {
        private string name;

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

        private double salary;

        public double Salary
        {
            get { return salary; }
            set { salary = value; }
        }

        private string position;

        public string Position
        {
            get { return position; }
            set { position = value; }
        }

        private string department;

        public string Department
        {
            get { return department; }
            set { department = value; }
        }

        private string email;

        public string Email
        {
            get { return email; }
            set { email = value; }
        }
```

```
        private int age;

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

    class Program
    {
        static void Main()
        {
            List<Employee> employees = new List<Employee>();

            int n = int.Parse(Console.ReadLine());
            while (n > 0)
            {
                var line = Console.ReadLine().Split().ToArray();

                int age = -1;
                string email = "n/a";
                if (line.Count() == 6)
                {
                    email = line[4];
                    age = int.Parse(line[5]);
                }
                else if (line.Count() > 4)
                {
                    if (!int.TryParse(line[4], out age))
                    {
                        age = -1;
                        if (!string.IsNullOrEmpty(line[4]))
                        {
                            email = line[4];
                        }
                    }
                }

                employees.Add
                (
                    new Employee()
                    {
                        // Required
                        Name = line[0],
                        Salary = double.Parse(line[1]),
                        Position = line[2],
                        Department = line[3],
                        // Optional
                        Email = email,
                        Age = age
                    }
                );
```

```
                n--;
            }

            Dictionary<string, List<double>> departmentsPartitioning = new
Dictionary<string, List<double>>();
            foreach (var employee in employees)
            {
                if (!departmentsPartitioning.ContainsKey(employee.Department))
                {
                    departmentsPartitioning[employee.Department] = new
List<double>();
                }
                departmentsPartitioning[employee.Department].Add(employee.Salary);
            }

            Dictionary<string, double> averageSalaryByDepartments = new
Dictionary<string, double>();
            foreach (var item in departmentsPartitioning.Keys)
            {
                averageSalaryByDepartments[item] =
departmentsPartitioning[item].Average();
            }

            var maxAverageSalaryDepartment =
averageSalaryByDepartments.OrderByDescending(item => item.Value).First();
            Console.WriteLine("Highest Average Salary: {0}",
maxAverageSalaryDepartment.Key);

            employees = employees.OrderByDescending(x => x.Salary).ToList();

             foreach (var employ in employees)
            {
                if (employ.Department == maxAverageSalaryDepartment.Key)
                {
                    Console.WriteLine($"{employ.Name} {employ.Salary} {employ.Email}
{employ.Age}");
                }
            }
        }
    }
}
```

Задача 2.2. Най-стария член на фамилията

Създайте клас Person с полета name и age. Създайте клас Family. Този клас трябва да има списък от хора, метод за добавяне на членове (void AddMember(Person member)) и метод, връщащ най-стария член на фамилията (Person GetOldestMember()). Напишете програма, която прочита името и възрастта на N души и ги добавя към фамилията. После отпечатва името и възрастта на най-стария ѝ член.

Бележки

Добавете в main метода следния код преди вашия. Ако сте дефинирали коректно класа, тестът би трябвало да мине успешно.

```
MethodInfo oldestMemberMethod = typeof(Family).GetMethod("GetOldestMember");
MethodInfo addMemberMethod = typeof(Family).GetMethod("AddMember");
if(oldestMemberMethod == null || addMemberMethod == null)
{
   throw new Exception();
}
```

Примери

Вход	Изход
3 Pesho 3 Gosho 4 Annie 5	Annie 5

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

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/228#2

Решение

```
namespace OldestFamilyMember
{
    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 override string ToString()
        {
            return $"Name: {this.name}, Age: {this.age}";
        }
    }

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

        public List<Person> People
        {
            get { return people; }
```

```
            set { people = value; }
        }

        public void AddMember(Person member)
        {
            people.Add(member);
        }

        public Person GetOldestMember()
        {
            return people.OrderByDescending(person => person.Age).FirstOrDefault();
        }
    }

    class Program
    {
        static void Main()
        {
            Family family = new Family();

            int n = int.Parse(Console.ReadLine());
            while (n > 0)
            {
                var line = Console.ReadLine().Split().ToArray();
                family.AddMember
                (
                    new Person()
                    {
                        Name = line[0],
                        Age = int.Parse(line[1])
                    }
                );
                n--;
            }

            Person oldestPerson = family.GetOldestMember();
            Console.WriteLine(oldestPerson.ToString());
        }
    }
}
```

Задача 2.3. Разликата в дни между две дати

Създайте клас DateModifier, който пресмята разликата в дни между две дати. Той трябва да съдържа метод, приемащ два низови параметъра, указващи дати в текстов формат и изчислява разликата в дни между тях.

Примери

Вход	Изход
1992 05 31 2016 06 17	8783
2016 05 31 2016 04 19	42

Тествайте решението: https://judge.softuni.bg/Contests/Practice/Index/228#4

Решение

```
namespace DatesDifference
{
    class DateModifier
    {
        private DateTime start;
        private DateTime end;

        public void SetDates(string firstDate, string secondDate)
        {
            var first = firstDate.Split().Select(int.Parse).ToArray();
            this.start = new DateTime(first[0], first[1], first[2]);

            var second = secondDate.Split().Select(int.Parse).ToArray();
            this.end = new DateTime(second[0], second[1], second[2]);
        }

        public int Difference()
        {
            int diff = (int)(end - start).TotalDays;
            return Math.Abs(diff);
        }
    }

    class Program
    {
        static void Main()
        {
            var firstDate = Console.ReadLine();
            var secondDate = Console.ReadLine();

            DateModifier date = new DateModifier();
            date.SetDates(firstDate, secondDate);

            Console.WriteLine(date.Difference());
        }
    }
}
```

Задача 2.4. Конструктори за класа Човек

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

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

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

Класът трябва да има и следните конструктори:

- **Person(string name, int age)**

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

Класът трябва да има и public метод за:

- **`GetBalance(): double`**

Добавете 2 конструктора към класа Person от миналата задача и с помощта на верижно извикване на кода използвайте повторно съществуващ вече програмен код:

1. Първият конструктор трябва да е без параметри и да създава човек с име "No name" и възраст = 1.
2. Вторият конструктор трябва да приема само един целочислен параметър за възрастта и да създава човек с име "No name" и възраст равна на подадения параметър.

По желание: Можете да се възползвате от верижното извикване на конструктори:

```
public Person(string name, int age)
    : this(name, age, new List<BankAccount>())
{ }

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

В класа трябва да присъства и конструктор, който приема низ за името и цяло число за възрастта и да създава личност с указаното име и възраст. Добавете следното към main метода и го качете в платформата.

```
Type personType = typeof(Person);
ConstructorInfo emptyCtor = personType.GetConstructor(new Type[] { });
ConstructorInfo ageCtor = personType.GetConstructor(new[] { typeof(int) });
ConstructorInfo nameAgeCtor = personType.GetConstructor(new[] { typeof(string),
typeof(int) });
bool swapped = false;
if (nameAgeCtor == null)
{
    nameAgeCtor = personType.GetConstructor(new[] { typeof(int), typeof(string) });
    swapped = true;
}

string name = Console.ReadLine();
```

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

Person basePerson = (Person)emptyCtor.Invoke(new object[] { });
Person personWithAge = (Person)ageCtor.Invoke(new object[] { age });
Person personWithAgeAndName = swapped ? (Person)nameAgeCtor.Invoke(new object[] {
age, name }) :(Person)nameAgeCtor.Invoke(new object[] { name, age });

Console.WriteLine("{0} {1}", basePerson.name, basePerson.age);
Console.WriteLine("{0} {1}", personWithAge.name, personWithAge.age);
Console.WriteLine("{0} {1}", personWithAgeAndName.name, personWithAgeAndName.age);
```

Ако сте дефинирали конструкторите коректно, тестът би трябвало да премине.

Примери

Вход	Изход
Pesho 20	No name 1 No name 20 Pesho 20
Gosho 18	No name 1 No name 18 Gosho 18
Stamat 43	No name 1 No name 43 Stamat 43

Решение

```
using System.Reflection;

namespace PersonConstructors
{
    class BankAccount
    {
        private int id;

        public int ID
        {
            get { return id; }
            set { id = value; }
        }

        private double balance;

        public double Balance
        {
            get { return balance; }
            set { balance = value; }
        }

        public BankAccount() : this(0, 0)
```

```
        {
            ;;
        }

        public BankAccount(int id, double balance)
        {
            this.id = id;
            this.balance = balance;
        }

        public void Deposit(double amount)
        {
            this.balance += amount;
        }

        public void Withdraw(double amount)
        {
            this.balance -= amount;
        }

        public override string ToString()
        {
            return $"Account {this.id}, balance {this.balance}";
        }
    }

    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; }
        }

        private List<BankAccount> accounts;

        public List<BankAccount> Accounts
        {
            get { return accounts; }
            set { accounts = value; }
        }

        public Person(string name, int age) : this(name, age, new
List<BankAccount>())
        {
```

```
            ; ;
        }

        public Person(string name, int age, List<BankAccount> accounts)
        {
            this.name = name;
            this.age = age;
            this.accounts = accounts;
        }

        public double GetBalance()
        {
            return this.accounts.Sum(item => item.Balance);
        }
    }

    class Program
    {
        static void Main()
        {
            Type personType = typeof(Person);
            ConstructorInfo emptyCtor = personType.GetConstructor(new Type[] { });
            ConstructorInfo ageCtor = personType.GetConstructor(new[] { typeof(int)
});
            ConstructorInfo nameAgeCtor = personType.GetConstructor(new[] {
typeof(string), typeof(int) });
            bool swapped = false;
            if (nameAgeCtor == null)
            {
                nameAgeCtor = personType.GetConstructor(new[] { typeof(int),
typeof(string) });
                swapped = true;
            }

            string name = Console.ReadLine();
            int age = int.Parse(Console.ReadLine());

            Person basePerson = (Person)emptyCtor.Invoke(new object[] { });
            Person personWithAge = (Person)ageCtor.Invoke(new object[] { age });
            Person personWithAgeAndName = swapped ? (Person)nameAgeCtor.Invoke(new
object[] { age, name }) : (Person)nameAgeCtor.Invoke(new object[] { name, age });

            Console.WriteLine("{0} {1}", basePerson.Name, basePerson.Age);
            Console.WriteLine("{0} {1}", personWithAge.Name, personWithAge.Age);
            Console.WriteLine("{0} {1}", personWithAgeAndName.Name,
personWithAgeAndName.Age);

        }
    }
}
```

Задача 2.5. Сурови данни

Вие сте собственик на куриерска компания и искате да направите система за проследяване на вашите коли и техния товар. Дефинирайте клас Car с информация за модела, двигателя, товара и колекция от точно 4

гуми. Моделът, товарът и гумите трябва да са отделни класове; създайте конструктор, който получава пълната информация за колата и създава и инициализира нейните вътрешни компоненти (двигател, товар и гуми).

На първия ред от входната информация ще получите число N - броя на колите, които имате, а на всеки от следващите N реда ще има информация за кола във формата "<Модел> <СкоростНаДвигателя> <МощностнаДвигателя> <ТеглоНаТовара> <ТипНаТовара> <Гума1Налягане> <Гума1Възраст> <Гума2Налягане> <Гума2Възраст> <Гума3Налягане> <Гума3Възраст> <Гума4Налягане> <Гума4Възраст>" където скорост, мощност, тегло на товара и възраст на гумите са цели числа, а налягането е дробно число, с двойна точност.

След тези N реда ще получите един-единствен ред с една от следните две команди: "fragile" или "flamable". Ако командата е "fragile", то отпечатайте всички коли с тип на товара "fragile" с гуми с налягане < 1; ако командата е "flamable", отпечатайте всички коли с тип на товара "flamable" и мощност на двигателя > 250. Колите трябва да се изведат в реда, в който са подадени като входни данни.

Примери

Вход	Изход
2 ChevroletAstro 200 180 1000 fragile 1.3 1 1.5 2 1.4 2 1.7 4 Citroen2CV 190 165 1200 fragile 0.9 3 0.85 2 0.95 2 1.1 1 fragile	Citroen2CV
4 ChevroletExpress 215 255 1200 flamable 2.5 1 2.4 2 2.7 1 2.8 1 ChevroletAstro 210 230 1000 flamable 2 1 1.9 2 1.7 3 2.1 1 DaciaDokker 230 275 1400 flamable 2.2 1 2.3 1 2.4 1 2 1 Citroen2CV 190 165 1200 fragile 0.8 3 0.85 2 0.7 5 0.95 2 flamable	ChevroletExpress DaciaDokker

Решение

```
using System;
using System.Reflection;

namespace RawData
{
    class Model
    {
        private string carModel;
```

```
        public string CarModel
        {
            get { return carModel; }
            set { carModel = value; }
        }

        private int engineSpeed;

        public int EngineSpeed
        {
            get { return engineSpeed; }
            set { engineSpeed = value; }
        }

        private int power;

        public int Power
        {
            get { return power; }
            set { power = value; }
        }
        public Model(string carModel, int engineSpeed, int power)
        {
            this.carModel = carModel;
            this.engineSpeed = engineSpeed;
            this.power = power;
        }
    }

    class Tovar
    {
        private int weight;

        public int Weight
        {
            get { return weight; }
            set { weight = value; }
        }

        private string type;

        public string Type
        {
            get { return type; }
            set { type = value; }
        }
        public Tovar(int weight, string type)
        {
            this.weight = weight;
            this.type = type;
        }
    }

    class Tyres
    {
```

```
        private int age;

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

        private double nalqgane;

        public double Nalqgane
        {
            get { return nalqgane; }
            set { nalqgane = value; }
        }

        public Tyres(double nalqgane, int age)
        {
            this.nalqgane = nalqgane;
            this.age = age;
        }
    }

    class Car
    {
        private Model carModel;

        public Model CarModel
        {
            get { return carModel; }
            set { carModel = value; }
        }
        private Tovar tovar;

        public Tovar Tovar
        {
            get { return tovar; }
            set { tovar = value; }
        }
        private Tyres[] carTyres = new Tyres[4];

        public Tyres[] CarTyres
        {
            get { return carTyres; }
            set { carTyres = value; }
        }

        public Car(Model model, Tovar tovar, Tyres[] tyres)
        {
            this.Tovar = tovar;
            this.carTyres = tyres;
            this.carModel = model;
        }
    }
```

```
    class Program
    {
        static void Main()
        {
            List<Car> cars = new List<Car>();

            int n = int.Parse(Console.ReadLine());
            while (n > 0)
            {
                var line = Console.ReadLine().Split().ToArray();
                var model = new Model(line[0], int.Parse(line[1]),
int.Parse(line[2]));
                var tovar = new Tovar(int.Parse(line[3]), line[4]);
                Tyres[] tyres = new Tyres[4];
                tyres[0] = new Tyres(double.Parse(line[5]), int.Parse(line[6]));
                tyres[1] = new Tyres(double.Parse(line[7]), int.Parse(line[8]));
                tyres[2] = new Tyres(double.Parse(line[9]), int.Parse(line[10]));
                tyres[3] = new Tyres(double.Parse(line[11]), int.Parse(line[12]));
                cars.Add(new Car(model, tovar, tyres));
                n--;
            }

            var type = Console.ReadLine();
            switch (type)
            {
                case "fragile":
                    var fragile = cars.Where(x => (x.CarTyres[0].Nalqgane < 1) &&
                                                  (x.CarTyres[1].Nalqgane < 1) &&
                                                  (x.CarTyres[2].Nalqgane < 1) &&
                                                  (x.CarTyres[3].Nalqgane < 1))
                                      .Select(y => y.CarModel.CarModel).ToList();
                    Console.WriteLine(string.Join("\n", fragile));
                    break;

                case "flamable":
                    var flamable = cars.Where(x => x.CarModel.Power > 250)
                                       .Select(y => y.CarModel.CarModel).ToList();
                    Console.WriteLine(string.Join("\n", flamable));
                    break;
            }
        }
    }
}
```

Задача 2.6. Пътувания с коли

Задачата ви е да напишете програма, която пази информация за автомобили, за това колко гориво имат и поддържа методи за движение на колите. Дефинирайте клас Car с информация за модела, количеството гориво, разхода на гориво за 1 км. и пропътуваното разстояние. Моделът на автомобилите е уникален - няма да има две коли с един и същи модел.

На първия ред на входните данни ще получите число N – броят на автомобилите, които ще следите. На всеки от следващите N реда ще има информация за по една кола в следния формат “<Модел>

<КоличествоГориво> <РазходНаГоривоЗа1км>". Всички коли започват с пропътувани 0 км.

След тези N реда, до достигане на команда "End", ще получавате команди във следния формат "Drive <МоделКола> <бройКм>". Реализирайте в класа Car метод, изчисляващ дали колата може да измине това разстояние или не. Ако да, горивото на колата трябва да бъде намалено с количеството на горивото, използвано за пътуването, а изминатите от нея километри трябва да бъдат увеличени с пропътуваните километри. Ако няма да може да го пропътува, колата не трябва да се движи (т.е. количеството на горивото и пропътуваните от нея километри трябва да останат същите), а на конзолата да се отпечата "Insufficient fuel for the drive". След достигане на команда "End" трябва да се отпечата всяка кола и нейното текущо количество гориво, както и пропътуваните километри във формата ""<Модел> <КоличествоГориво> <пропътуваниКм>", където количеството гориво трябва да е отпечатано с две цифри след десетичния знак.

Примери

Вход	Изход
2 AudiA4 23 0.3 BMW-M2 45 0.42 Drive BMW-M2 56 Drive AudiA4 5 Drive AudiA4 13 End	AudiA4 17.60 18 BMW-M2 21.48 56
3 AudiA4 18 0.34 BMW-M2 33 0.41 Ferrari-488Spider 50 0.47 Drive Ferrari-488Spider 97 Drive Ferrari-488Spider 35 Drive AudiA4 85 Drive AudiA4 50 End	Insufficient fuel for the drive Insufficient fuel for the drive AudiA4 1.00 50 BMW-M2 33.00 0 Ferrari-488Spider 4.41 97

Решение

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

```
        private float fuel;
        public float Fuel
        {
            get { return fuel; }
            set { fuel = value; }
        }

        private float perkm;
        public float Perkm
        {
            get { return perkm; }
            set { perkm = value; }
        }

        public void TheMagic(float km)
        {
            if (km * this.perkm > this.fuel) Console.WriteLine("Insufficient fuel
for the drive");
            else
            {
                this.fuel -= (km * this.perkm);
                Console.WriteLine("{0} {1:f2} {2}", this.name, this.fuel, km);
            }
        }
    }

    class Program
    {
        static void Main()
        {
            int n = int.Parse(Console.ReadLine());
            List<Car> cars = new List<Car>();
            for (int i = 0; i < n; i++)
            {
                var fuk = Console.ReadLine().Split();
                cars.Add
                (
                    new Car()
                    {
                        Name = fuk[0],
                        Fuel = float.Parse(fuk[1]),
                        Perkm = float.Parse(fuk[2])
                    }
                );
            }

            var items = new Dictionary<string, int>();
            var line = "";
            do
            {
                line = Console.ReadLine();
                if (line != "End")
                {
                    var split = line.Split();
```

```
                if (items.ContainsKey(split[1]))
                {
                    var km = items[split[1]] + int.Parse(split[2]);
                    items[split[1]] = km;

                }
                else items.Add(split[1], int.Parse(split[2]));
            }
        }
        while (line != "End");

        foreach (var item in items)
        {
            var currentCar = cars.Where(a => a.Name == item.Key).First();
            currentCar.TheMagic(item.Value);
        }
    }
}
```

Задача 2.7. Застъпване на правоъгълници

Създайте клас Rectangle. Той трябва да съдържа полета ID, широчина, височина и координатите на неговия горен ляв ъгъл (по хоризонтала и по вертикала). Създайте метод, който получава като параметър друг обект Rectangle, проверява дали двата правоъгълника се застъпват и връща true или false.

На първия ред ще получите броя правоъгълници – N и броя проверки за застъпване – M. На следващите N ще получавате правоъгълници с тяхното ID, широчина, височина и координати. На последните M реда ще са двойки от ID-та на правоъгълници. Отпечатате дали при всяка от тези двойки има застъпване.

Винаги ще получавате валидни данни. Няма нужда да проверявате дали правоъгълниците съществуват.

Примери

Вход	Изход
2 1 Pesho 2 2 0 0 Gosho 2 2 0 0 Pesho Gosho	true

Решение

```
namespace RectanglesOverlap
{
    class Rectangle
    {
        private string id;
        public string ID
        {
            get { return id; }
```

```
            set { id = value; }
        }

        private int width;
        public int Width
        {
            get { return width; }
            set { width = value; }
        }

        private int heigh;
        public int Heigh
        {
            get { return heigh; }
            set { heigh = value; }
        }

        private int horizontally;
        public int Horizontally
        {
            get { return horizontally; }
            set { horizontally = value; }
        }

        private int vertically;
        public int Vertically
        {
            get { return vertically; }
            set { vertically = value; }
        }
    }

    class Program
    {
        static void Main()
        {
            List<Rectangle> info = new List<Rectangle>();

            int[] rectangles = Console.ReadLine().Split('
').Select(int.Parse).ToArray();
            for (int i = 0; i < rectangles[0]; i++)
            {
                string[] recInfo = Console.ReadLine().Split(' ').ToArray();
                info.Add
                (
                    new Rectangle()
                    {
                        ID = recInfo[0],
                        Width = int.Parse(recInfo[1]),
                        Heigh = int.Parse(recInfo[2]),
                        Horizontally = int.Parse(recInfo[3]),
                        Vertically = int.Parse(recInfo[4])
                    }
                );
            }
```

```
            string[] couples = Console.ReadLine().Split(' ');
            var first = info.Where(a => a.ID == couples[0]).First();
            var second = info.Where(a => a.ID == couples[1]).First();
            if (first.Width == second.Width && first.Heigh == second.Heigh &&
first.Horizontally == second.Horizontally && first.Vertically == second.Vertically)
            {
                Console.WriteLine("true");
            }
            else Console.WriteLine("false");
        }
    }
}
```

Задача 2.8. Продавач на коли

Дефинирайте два класа Car и Engine. Класът за колите Car има полета за модел, двигател, тегло и цвят (model, engine, weight, color). Всеки двигател (Engine) има характеристики модел, мощност, кубатура и КПД (model, power, displacement, efficiency). Теглото и цвета на колата и кубатурата и КПД-то на двигателя ѝ са незадължителни данни.

На първия ред ще получите число N, показващо колко реда с информации за двигатели ще получите, на всеки от следващите N реда ще има информация за по един двигател в следния формат "<Model> <Power> <Displacement> <Efficiency>". След редовете с двигателите, на следващия ред ще получите число M – указващо броя на колите, които следват. На всеки от следващите M реда ще има информация за една кола в следния формат "<Model> <Engine> <Weight> <Color>", където двигателят ще е модел на съществуващ (описан вече) двигател. Когато създавате обект за кола, трябва да пазите указател към точния двигател, вместо само модела на двигателя. Обърнете внимание на това, че незадължителните характеристики може да липсват от форматираните данни.

Вашата задача е да отпечатате всяка кола (в реда, в който сте ги получили) и информацията за нея във вида, посочен по-долу; ако някое от незадължителните полета не е посочено, отпечатайте "n/a" на негово място:

```
<CarModel>:
  <EngineModel>:
    Power: <EnginePower>
    Displacement: <EngineDisplacement>
    Efficiency: <EngineEfficiency>
  Weight: <CarWeight>
  Color: <CarColor>
```

Бонус

Предефинирайте методите ToString(), така че да имате многократно използваем начин за извеждане на тези обекти.

Примери

Вход	Изход

2
V8-101 220 50
V4-33 140 28 B
3
FordFocus V4-33 1300 Silver
FordMustang V8-101
VolkswagenGolf V4-33 Orange
	FordFocus:
	  V4-33:
	  Power: 140
	  Displacement: 28
	  Efficiency: B
	  Weight: 1300
	  Color: Silver
	FordMustang:
	  V8-101:
	  Power: 220
	  Displacement: 50
	  Efficiency: n/a
	  Weight: n/a
	  Color: n/a
	VolkswagenGolf:
	  V4-33:
	  Power: 140
	  Displacement: 28
	  Efficiency: B
	  Weight: n/a
	  Color: Orange

4
DSL-10 280 B
V7-55 200 35
DSL-13 305 55 A+
V7-54 190 30 D
4
FordMondeo DSL-13 Purple
VolkswagenPolo V7-54 1200 Yellow
VolkswagenPassat DSL-10 1375 Blue
FordFusion DSL-13
	FordMondeo:
	  DSL-13:
	  Power: 305
	  Displacement: 55
	  Efficiency: A+
	  Weight: n/a
	  Color: Purple
	VolkswagenPolo:
	  V7-54:
	  Power: 190
	  Displacement: 30
	  Efficiency: D
	  Weight: 1200
	  Color: Yellow
	VolkswagenPassat:
	  DSL-10:
	  Power: 280
	  Displacement: n/a
	  Efficiency: B
	  Weight: 1375
	  Color: Blue
	FordFusion:

	DSL-13:
	  Power: 305
	  Displacement: 55
	  Efficiency: A+
	Weight: n/a
	Color: n/a

Решение

Задача 2.9. Треньор на покемони

Вие искате да сте най-добрия треньор на покемони, по-добър от всеки друг, така че сте се заели да ловите покемони. Дефинирайте клас Trainer и клас Pokemon. Треньорът трябва да има име, брой значки и колекция от покемони. Покемонът има име, елемент и здраве, всички стойности са задължителни. Всеки треньор започва с 0 значки.

От конзолата ще получите неизвестно колко редове, след които ще следва команда "Tournament". Всеки от тези редове ще носи информация за покемона и треньора, който го е хванал във формата "<ИмеНаТреньор> <ИмеНаПокемон> <ЕлементНаПокемона> <ЗдравеНаПокемона>" където ИмеНаТреньор е името на треньора, хванал покемона; имената са уникални, няма как да има двама треньори с еднакви имена. След получаване на команда "Tournament" неизвестен брой редове ще съдържат като команда един от трите елемента "Fire", "Water", "Electricity", които продължават, докато се получи команда "End". За всяка от тези команди трябва да проверите дали треньорът има поне един покемон с дадения елемент. Ако да, треньорът получава 1 значка, в противен случай всичките му покемони губят 10 точки здраве, а ако даден покемон падне до 0 или по-малко точки здраве той умира и трябва да бъде изтрит от колекцията на треньора. След като бъде получена команда "End" трябва да отпечатите всички треньори, сортирани според броя на значките, които имат, в намаляващ ред (ако двама треньори имат еднакъв брой значки те трябва да са сортирани според реда на тяхното появяване във входните данни), във формата "<ИмеНаТреньор> <Значки> <БройПокемони>".

Примери

Вход	Изход
Pesho Charizard Fire 100 Gosho Squirtle Water 38 Pesho Pikachu Electricity 10 Tournament Fire Electricity End	Pesho 2 2 Gosho 0 1
Stamat Blastoise Water 18	Nasko 1 1

Nasko Pikachu Electricity 22 Jicata Kadabra Psychic 90 Tournament Fire Electricity Fire End	Stamat 0 0 Jicata 0 1

Решение

Задача 2.10. Google

Google винаги ви наблюдава, така че не би трябвало да сте изненадани, че те знаят всичко за вас (дори и за вашата покемон колекция). И понеже вие сте наистина добри в писането на класове от Google са ви помолили да напишете клас, който съдържа цялата информация, която те искат да събират за хората.

От конзолата ще получите неясно колко редове, завършващи накрая с команда "End". На всеки от тези редове ще е информацията за един човек в един от следните формати:

- "<Име> company <имеНаФирма> <отдел> <заплата>"
- "<Име> pokemon <имеНаПокемон> <видНаПокемона>"
- "<Име> parents <имеНаРодител> <рожденДенНаРодителя>"
- "<Име> children <имеНаДете> <рожденДенНаДете>"
- "<Име> car <моделНаКолата> <скоростНаКолата>"

Вие трябва да структурирате цялата информация за всеки човек в клас с вложени класове. Имената на хората са уникални - няма двама души с еднакви имена, човек може да има само 1 фирма и кола, но има множество родители, деца и покемони. След като се получи команда "End" на следващия ред ще получите едно име и трябва да отпечатите цялата информация за този човек. Имайте в предвид, че информацията може да се промени във входните данни, например ако се получат множество редове, които указват фирмата на човека, само последния е този, който трябва да бъде запомнен. Заплатата трябва да бъде изведена с два знака след десетичния разделител.

Примери

Вход	Изход
PeshoPeshev company PeshInc Management 1000.00 TonchoTonchev car Trabant 30 PeshoPeshev pokemon Pikachu Electricity PeshoPeshev parents PoshoPeshev 22/02/1920 TonchoTonchev pokemon Electrode Electricity	TonchoTonchev Company: Car: Trabant 30 Pokemon: Electrode Electricity

End TonchoTonchev	Parents: Children:
JelioJelev pokemon Onyx Rock JelioJelev parents JeleJelev 13/03/1933 GoshoGoshev pokemon Moltres Fire JelioJelev company JeleInc Jelior 777.77 JelioJelev children PudingJelev 01/01/2001 StamatStamatov pokemon Blastoise Water JelioJelev car AudiA4 180 JelioJelev pokemon Charizard Fire End JelioJelev	JelioJelev Company: JeleInc Jelior 777.77 Car: AudiA4 180 Pokemon: Onyx Rock Charizard Fire Parents: JeleJelev 13/03/1933 Children: PudingJelev 01/01/2001

Бонус

Предефинирайте метода ToString() в дефинираните класове, за да стандартизирате извеждането на обектите.

Решение

```
namespace Google
{
    class Company
    {
        private string name;
        private string department;
        private double salary;

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

        public string Department
        {
            get { return department; }
            set { department = value; }
        }

        public double Salary
        {
            get { return salary; }
            set { salary = value; }
        }

        public override string ToString()
        {
            if (name == null) return string.Empty;
            else return $"{name} {department} {salary}\n";
```

```
        }
    }

    class Car
    {
        private string model;
        private int speed;

        public string Model
        {
            get { return model; }
            set { model = value; }
        }

        public int Speed
        {
            get { return speed; }
            set { speed = value; }
        }

        public override string ToString()
        {
            return $"{model} {speed}\n";
        }
    }

    class Child
    {
        private string name;
        private string birthday;

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

        public string Birthday
        {
            get { return birthday; }
            set { birthday = value; }
        }
        public override string ToString()
        {
            return $"{name} {birthday}";
        }
    }

    class Parent
    {
        private string name;
        private string birthday;

        public string Name
        {
```

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

    public string Birthday
    {
        get { return birthday; }
        set { birthday = value; }
    }

    public override string ToString()
    {
        return $"{name} {birthday}";
    }
}

class Person
{
    private Company company;
    private Car car;
    private List<Pokemon> pokemons;
    private List<Child> children;
    private List<Parent> parents;

    public Company Company
    {
        get { return company; }
        set { company = value; }
    }
    public Car Car
    {
        get { return car; }
        set { car = value; }
    }
    public List<Pokemon> Pokemons
    {
        get { return pokemons; }
        set { pokemons = value; }
    }
    public List<Child> Children
    {
        get { return children; }
        set { children = value; }
    }
    public List<Parent> Parents
    {
        get { return parents; }
        set { parents = value; }
    }
}

class Pokemon
{
    private string name;
    private string type;
```

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

        public string Type
        {
            get { return type; }
            set { type = value; }
        }
        public override string ToString()
        {
            return $"{name} {type}";
        }
    }

    class Program
    {
        static void SetCompany(Dictionary<string, Person> people, string name,
string nameOfCompany, string department, double salary)
        {
            people[name].Company.Name = nameOfCompany;
            people[name].Company.Department = department;
            people[name].Company.Salary = salary;
        }

        static void SetCar(Dictionary<string, Person> people, string name, string
model, int speed)
        {
            people[name].Car.Model = model;
            people[name].Car.Speed = speed;
        }

        static void SetPokemon(Dictionary<string, Person> people, string name,
string nameOfPokemon, string type)
        {
            people[name].Pokemons.Add(new Pokemon { Name = nameOfPokemon, Type =
type });
        }

        static void SetParents(Dictionary<string, Person> people, string name,
string nameOfParent, string birthday)
        {
            people[name].Parents.Add(new Parent { Name = nameOfParent, Birthday =
birthday });
        }

        static void SetChildren(Dictionary<string, Person> people, string name,
string nameOfChild, string birthday)
        {
            people[name].Children.Add(new Child { Name = nameOfChild, Birthday =
birthday });
        }
```

```
        static void Output(Dictionary<string, Person> people, string name)
        {
            Console.WriteLine(name);
            Console.WriteLine("Company:");
            Console.Write(people[name].Company.ToString());
            Console.WriteLine("Car:");
            Console.Write(people[name].Car.ToString());
            Console.WriteLine("Pokemon:");
            foreach (var pokemon in people[name].Pokemons)
            {
                Console.WriteLine(pokemon.ToString());
            }
            Console.WriteLine("Parents:");
            foreach (var parent in people[name].Parents)
            {
                Console.WriteLine(parent.ToString());
            }
            Console.WriteLine("Children:");
            foreach (var child in people[name].Children)
            {
                Console.WriteLine(child.ToString());
            }
        }

        static void Main(string[] args)
        {
            Dictionary<string, Person> people = new Dictionary<string, Person>();
            string[] line = Console.ReadLine().Split(' ').ToArray();
            while (line[0] != "End")
            {
                string command = line[1];
                string namenew = line[0];
                if (!people.ContainsKey(namenew))
                {
                    people[namenew] = new Person();
                    people[namenew].Pokemons = new List<Pokemon>();
                    people[namenew].Parents = new List<Parent>();
                    people[namenew].Children = new List<Child>();
                    people[namenew].Car = new Car();
                    people[namenew].Company = new Company();
                }
                switch (command)
                {
                    case "company":
                        SetCompany(people, line[0], line[2], line[3],
double.Parse(line[4]));
                        break;
                    case "parents":
                        SetParents(people, line[0], line[2], line[3]);
                        break;
                    case "children":
                        SetChildren(people, line[0], line[2], line[3]);
                        break;
                    case "pokemon":
```

```
                    SetPokemon(people, line[0], line[2], line[3]);
                    break;
                case "car":
                    SetCar(people, line[0], line[2], int.Parse(line[3]));
                    break;
            }
            line = Console.ReadLine().Split(' ').ToArray();
        }
        string name = Console.ReadLine();
        Output(people, name);
    }
}
}
```

Задача 2.11. Родословно дърво

Решили сте да направите родословно дърво, така че сте поразпитали баба си за фамилията. За съжаление тя помни само откъслечна информация за предците ви, затова на вас се пада честта да обобщите информацията и да построите родословното дърво.

На първия ред на входните данни ще получите или име, или дата на раждане във формати "<Име> <Фамилия>" или "ден/месец/година". Вашата задача ще бъде да откриете информацията за човека в родословното дърво. На следващите редове до команда "End" ще получавате информация за вашите предци, която ще ви е нужна за построяване на фамилното дърво.

Информацията ще бъде в един от следните формати:

- "Име Фамилия - Име Фамилия"
- "Име Фамилия - ден/месец/година"
- "ден/месец/година - Име Фамилия"
- "ден/месец/година - ден/месец/година"
- "Име Фамилия ден/месец/година"

Първите 4 формата разкриват семейна връзка – лицето отляво е родител на лицето отдясно (както виждате, не е задължително форматът да съдържа имена, например 4-тия формат означава, че лицето, родено на датата отляво е родител на лицето, родено на датата отдясно). Последният формат свързва друг тип информация - например лицето с това и това име е родено на тази и тази дата. Имената и рожденните дати са уникални – няма да има двама души със съвпадащо име или рождена дата, винаги ще има достатъчно данни за да се състави родословното дърво (имената и рожденните дати на всички хора са известни и всеки от тях ще има поне една връзка с някой друг от родословното дърво).

След получаването на команда "End" трябва да отпечатите цялата информация за лицето, чието име или рождена дата сте получили на първия ред – неговото име, рожден ден, родители и деца (проверете

примерите за изисквания формат). Хората в списъка на родителите и децата трябва да бъдат подредени според тяхното първо появяване във входните данни (без значение дали са били подадени като рожденна дата или като име - например в първата серия примерни данни Стамат е преди Пенка, защото той е споменат пръв на втория ред, докато тя се появява за пръв път на третия).

Примери

Вход	Изход
Pesho Peshev 11/11/1951 - 23/5/1980 Penka Pesheva - 23/5/1980 Penka Pesheva 9/2/1953 Pesho Peshev - Gancho Peshev Gancho Peshev 1/1/2005 Stamat Peshev 11/11/1951 Pesho Peshev 23/5/1980 End	Pesho Peshev 23/5/1980 Parents: Stamat Peshev 11/11/1951 Penka Pesheva 9/2/1953 Children: Gancho Peshev 1/1/2005
13/12/1993 25/3/1934 - 4/4/1961 Poncho Tonchev 25/3/1934 4/4/1961 - Moncho Tonchev Toncho Tonchev - Lomcho Tonchev Moncho Tonchev 13/12/1993 Lomcho Tonchev 7/7/1995 Toncho Tonchev 4/4/1961 End	Moncho Tonchev 13/12/1993 Parents: Toncho Tonchev 4/4/1961 Children:

Решение

```
using System.Globalization;

namespace FamilyTree
{
    class FamilyInfo
    {
        public List<Person> Parents { get; set; }

        public List<Person> Children { get; set; }

        public FamilyInfo()
        {
            this.Parents = new List<Person>();
            this.Children = new List<Person>();
        }
    }

    class Person
    {
        public string Name { get; set; }
```

```
        public DateTime BirthDate { get; set; }

        public FamilyInfo FamilyInfo { get; set; }

        public Person()
        {
            this.FamilyInfo = new FamilyInfo();
        }

        public Person(string name) : this()
        {
            this.Name = name;
        }

        public Person(DateTime birthDate) : this()
        {
            this.BirthDate = birthDate;
        }

        public Person(string name, DateTime birthDate)
            : this(name)
        {
            this.BirthDate = birthDate;
        }

        public FamilyInfo GetFamilyInfo()
        {
            return this.FamilyInfo;
        }

        public override string ToString()
        {
            return $"{this.Name} {this.BirthDate.ToString("d/M/yyyy")}";
        }
    }

    class FamilyTree
    {
        public static List<Person> FamilyMembers { get; set; }

        public static Person TargetMember { get; private set; }

        public static void Create()
        {
            FamilyMembers = new List<Person>();
        }

        public static void SetTargetMember(string targetPerson)
        {
            TargetMember = GetPerson(targetPerson);
        }

        public static void AddMember(Person familyMember)
        {
```

```
            FamilyMembers.Add(familyMember);
        }

        public static void AddRelation(Person parent, Person child)
        {
            parent.FamilyInfo.Children.Add(child);
            child.FamilyInfo.Parents.Add(parent);
        }

        public static void AddRelation(string relation)
        {
            var p1Info = relation.Split('-').Select(p => p.Trim()).First();
            var p2Info = relation.Split('-').Select(p => p.Trim()).Last();

            Person person1 = GetPerson(p1Info);
            Person person2 = GetPerson(p2Info);

            AddRelation(person1, person2);

        }

        private static Person GetPerson(string personInfo)
        {
            Person person;

            var isInfoDate = DateTime.TryParseExact(personInfo, "d/M/yyyy",
CultureInfo.InvariantCulture,
                DateTimeStyles.None, out DateTime personBirthDate);

            if (!isInfoDate)
                person = FamilyMembers.Where(p => p.Name == personInfo).First();
            else
                person = FamilyMembers.Where(p => p.BirthDate ==
personBirthDate).First();

            return person;
        }
    }

    class Program
    {
        public static void Main(string[] args)
        {
            var targetPerson = Console.ReadLine();

            FamilyTree.Create();

            var information = new List<string>();
            string info;

            while ((info = Console.ReadLine()) != "End")
            {
                information.Add(info);
            }
```

```
            var newMembers = new List<string>(information.Where(s => !s.Contains('-
')));

            foreach (var member in newMembers)
            {
                var parts = member.Split(' ');

                var birthDate = DateTime.ParseExact(parts.Last(), "d/M/yyyy",
                    CultureInfo.InvariantCulture);

                var name = string.Join(" ", parts.Take(parts.Length - 1));

                FamilyTree.AddMember(new Person(name, birthDate));
            }

            FamilyTree.SetTargetMember(targetPerson);

            var relations = new List<string>(information.Where(s => s.Contains('-
')));

            foreach (var relation in relations)
            {
                FamilyTree.AddRelation(relation);
            }

            Console.WriteLine(FamilyTree.TargetMember);
            Console.WriteLine("Parents:");
            Console.WriteLine(String.Join(Environment.NewLine,
FamilyTree.TargetMember.FamilyInfo.Parents));
            Console.WriteLine("Children");
            Console.WriteLine(String.Join(Environment.NewLine,
FamilyTree.TargetMember.FamilyInfo.Children));
        }
    }
}
```

Тема 3. Енкапсулация на данни

Задача 3.1. Сортиране на хора по име и възраст

Създайте class Person, който да има private полета:

- firstName: string
- lastName: string
- age: int
- ToString(): string - override

Примери

Input	Output

5 Asen Ivanov 65 Boiko Borisov 57 Ventsislav Ivanov 27 Asen Harizanoov 44 Boiko Angelov 35	Asen Harizanoov is a 44 years old Asen Ivanov is a 65 years old Boiko Angelov is a 35 years old Boiko Borisov is a 57 years old Ventsislav Ivanov is a 27 years old

Решение

```
namespace PersonSort
{
    class Person
    {
        private string firstName;

        public string FirstName
        {
            get { return firstName; }
            set { firstName = value; }
        }

        private string lastName;

        public string LastName
        {
            get { return lastName; }
            set { lastName = value; }
        }

        private int age;

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

        public override string ToString()
        {
            return $"{this.FirstName} {this.LastName} is a {this.age} years old";
        }

        public Person(string firstName, string lastName)
        {
            this.firstName = firstName;
            this.lastName = lastName;
        }

        public Person(string firstName, string lastName, int age) : this(firstName,
lastName)
        {
            this.age = age;
        }
    }
```

```
    class Program
    {
        public static void Main()
        {
            var lines = int.Parse(Console.ReadLine());
            var persons = new List<Person>();
            for (int i = 0; i < lines; i++)
            {
                var cmdArgs = Console.ReadLine().Split();
                var person = new Person(cmdArgs[0], cmdArgs[1],
int.Parse(cmdArgs[2]));
                persons.Add(person);
            }

            persons.OrderBy(p => p.FirstName)
                   .ThenBy(p => p.Age)
                   .ToList()
                   .ForEach(p => Console.WriteLine(p.ToString()));
        }
    }
}
```

Задача 3.2. Клас Box (правоъгълен паралелепипед)

Дадена е геометричната фигура box с параметри дължина, широчина и височина. Направете клас Box, който да се инстанцира по тези параметри. Дайте на външния свят само методите за лице на повърхнина, околна повърхнина и обем (Формулите може да намерите на адрес: http://www.mathwords.com/r/rectangular_parallelepiped.htm).

На първите три реда ще получите дължина, ширина и височина.

На следващите три реда се извеждат повърхнината, околната повърхнина и обема на паралелепипеда:

Забележка

Добавете следващия код в началото на метода main.

```
static void Main(string[] args)
{
    Type boxType = typeof(Box);
    FieldInfo[]     fields     =     boxType.GetFields(BindingFlags.NonPublic     |
BindingFlags.Instance);
    Console.WriteLine(fields.Count());
}
```

Ако сте дефинирали коректно класа, теста ще мине.

Примери

Вход	Изход

2 3 4	3 Surface Area – 52.00 Lateral Surface Area – 40.00 Volume – 24.00
1.3 1 6	3 Surface Area - 30.20 Lateral Surface Area - 27.60 Volume - 7.80

Решение

```
namespace BoxClass
{
    class Box
    {
        private float lenght;

        private float width;

        private float height;

        public Box(float lenght, float width, float height)
        {
            this.lenght = lenght;
            this.width = width;
            this.height = height;
        }

        public float SurfaceArea()
        {
            return 2 * this.lenght * this.width +
                   2 * this.lenght * this.height +
                   2 * this.width * this.height;
        }

        public float LateralSurfaceArea()
        {
            return 2 * this.lenght * this.height +
                   2 * this.width * this.height;
        }

        public float Volume()
        {
            return this.lenght * this.width * this.height;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            //// Reflection Test
            // Type boxType = typeof(Box);
```

```
            // FieldInfo[] fields = boxType.GetFields(BindingFlags.NonPublic |
BindingFlags.Instance);
            // Console.WriteLine("Non Public Fields: " + fields.Count());

            float lenght = float.Parse(Console.ReadLine());
            float width = float.Parse(Console.ReadLine());
            float height = float.Parse(Console.ReadLine());

            Box box = new Box(lenght, width, height);

            Console.WriteLine("Surface Area - {0:f2}", box.SurfaceArea());
            Console.WriteLine("Lateral Surface Area - {0:f2}",
box.LateralSurfaceArea());
            Console.WriteLine("Volume - {0:f2}", box.Volume());
        }
    }
}
```

Задача 3.3. Увеличение на заплатата

Преструктурирайте (рефактурирайте) проекта с клас Person.

Въвеждаме хора (Person) с техните имена, възраст и заплата. Въвеждаме процент бонус към заплатата на всеки обект person. Обектите Persons, на възраст под 30 получават бонус на половина. Разширяваме Person от предишната задача. Нови полета и методи:

- salary: double
- IncreaseSalary(double bonus)

Примери

Вход	Изход
5 Asen Ivanov 65 2200 Boiko Borisov 57 3333 Ventsislav Ivanov 27 600 Asen Harizanoov 44 666.66 Boiko Angelov 35 559.4 20	Asen Ivanov get 2640.00 leva Boiko Borisov get 3999.60 leva Ventsislav Ivanov get 660.00 leva Asen Harizanoov get 799.99 leva Boiko Angelov get 671.28 leva

Решение

```
namespace IncreaseSalary
{
    class Person
    {
        private string firstName;

        public string FirstName
        {
            get { return firstName; }
            set { firstName = value; }
        }
```

```
private string lastName;

public string LastName
{
    get { return lastName; }
    set { lastName = value; }
}

private int age;

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

private double salary;

public double Salary
{
    get { return salary; }
    set { salary = value; }
}

public void IncreaseSalary(double bonus)
{
    if (this.age > 30)
    {
        this.salary += this.salary * bonus / 100;
    }
    else
    {
        this.salary += this.salary * bonus / 200;
    }
}

public override string ToString()
{
    return $"{this.FirstName} {this.LastName} get {this.salary:f2} leva";
}

public Person(string firstName, string lastName)
{
    this.firstName = firstName;
    this.lastName = lastName;
}

public Person(string firstName, string lastName, int age)
      : this(firstName, lastName)
{
    this.age = age;
}

public Person(string firstName, string lastName, int age, double salary)
```

```
            : this(firstName, lastName, age)
        {
            this.salary = salary;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var lines = int.Parse(Console.ReadLine());
            var persons = new List<Person>();
            for (int i = 0; i < lines; i++)
            {
                var cmdArgs = Console.ReadLine().Split();
                var person = new Person
                (
                    cmdArgs[0],
                    cmdArgs[1],
                    int.Parse(cmdArgs[2]),
                    double.Parse(cmdArgs[3])
                );
                persons.Add(person);
            }

            var bonus = double.Parse(Console.ReadLine());
            persons.ForEach(person => person.IncreaseSalary(bonus));

            persons.ForEach(p => Console.WriteLine(p.ToString()));

        }
    }
}
```

Задача 3.4. Ферма за животни

Трябва да сте запознати с капсулирането вече. За тази задача ще се работи по проект Животинска ферма. Можете да го откриете в AnimalFarm.zip. Той съдържа клас Chicken. Добавете към него няколко полета, конструктор, свойства и метода по ваша преценка, така че да може да се ползва класа по указания в примерите начин. Вашата задача освен това ще бъде да се форматират или скрият членовете на класа, които не са предназначени да се виждат или модифицират извън класа.

Step 1. Капсулиране на полетата

Полетата трябва да бъдат private. Оставянето на полетата, отворени за промяна извън класа, е потенциално опасно. Направете всички полета в класа Chicken private. В случай, че стойността на полето е необходимо другаде, използвайте getters за достъп до него.

Step 2. Подходящо валидиране на данните (вижте как в упражнението за валидация)

Валидирайте името на обектите от клас Chiken (не може да е null, празно или да съдържа само интервали). В случай на невалидно име, да се върне изключение със следното съобщение "Name cannot be empty."

Валидирайте свойството age, с минималната и максималната допустими стойности. В случай на невалидна възраст, върнете изключение със съобщение "Age should be between 0 and 15."

Не забравяйте да обработвате правилно вероятно получените изключения.

Step 3. Направете класове да имат валидно начално състояние

Наличието на getters и setters е безполезно, ако всъщност не ги използвате. Конструкторът на класа Chicken променя полетата директно, което е погрешно, когато са налице подходящи механизми за валидиране на входните данни. Променете конструктора за да разрешите този проблем.

Step 4. Скрийте вътрешната логика

Ако метод е предназначен да се използва само от наследяващите класове или вътрешно да извърши някакво действие, няма смисъл да бъде публичен. Методът CalculateProductPerDay() се използва от productPerDay(), който е public getter. Това означава, че методът безопасно може да бъде скрит във вътрешността на класа Chicken като се декларира като private.

Примери

Вход	Изход
Mara 10	Chicken Mara (age 10) can produce 1 eggs per day.
Mara 17	Age should be between 0 and 15.

Решение

```
namespace AnimalFarm
{
    class Chicken
    {
        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 Chicken(string name, int age)
        {
            if (string.IsNullOrEmpty(name) || string.IsNullOrWhiteSpace(name))
            {
                throw new ArgumentException("Name cannot be empty.");
            }

            this.name = name;
            this.age = age;
        }

        public string CalculateProductPerDay()
        {
            if (this.age < 0 || this.age > 15)
            {
                throw new ArgumentException("Age should be between 0 and 15.");
            }

            return $"Chicken {this.name}(age {this.age}) can produce 1 eggs per
day.";
        }
    }

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

            Chicken chicken = new Chicken(name, age);

            Console.WriteLine(chicken.CalculateProductPerDay());
        }
    }
}
```

Задача 3.5. Проверка на данните

Разширяваме класа Person с подходяща валидация за всяко поле:

- Имената трябва да са поне 3 символа
- възрастта не трябва да е нула или отрицателно число
- заплатата не може да бъде по-малка от 460.0

Print proper message to end user (look at example for messages).

Use ArgumentExeption with messages from example.

Изведете подходящо съобщение за последен потребител (виж примера за съобщения).

Използвайте ArgumentExeption със съобщения от примера.

Примери

Вход	Изход
5 Asen Ivanov -6 2200 B Borisov 57 3333 Ventsislav Ivanov 27 600 Asen H 44 666.66 Boiko Angelov 35 300 20	Age cannot be zero or negative integer First name cannot be less than 3 symbols Last name cannot be less than 3 symbols Salary cannot be less than 460 leva Ventsislav Ivanov get 660.0 leva

Решение

```
namespace PersonDataValidation
{
    class Person
    {
        private string firstName;

        public string FirstName
        {
            get { return firstName; }
            set
            {
                if (value.Length < 3)
                {
                    throw new ArgumentException("First name cannot be less than 3
symbols");
                }
                firstName = value;
            }
        }

        private string lastName;

        public string LastName
        {
            get { return lastName; }
            set
            {
                if (value.Length < 3)
                {
                    throw new ArgumentException("Last name cannot be less than 3
symbols");
                }
                lastName = value;
            }
        }
```

```
        private int age;

        public int Age
        {
            get { return age; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Age cannot be zero or negative
integer");
                }
                age = value;
            }
        }

        private double salary;

        public double Salary
        {
            get { return salary; }
            set
            {
                if (value < 460)
                {
                    throw new ArgumentException("Salary cannot be less than 460
leva");
                }
                salary = value;
            }
        }

        public void IncreaseSalary(double bonus)
        {
            if (this.age > 30)
            {
                this.salary += this.salary * bonus / 100;
            }
            else
            {
                this.salary += this.salary * bonus / 200;
            }
        }

        public override string ToString()
        {
            return $"{this.FirstName} {this.LastName} get {this.salary:f2} leva";
        }

        public Person(string firstName, string lastName)
        {
            this.FirstName = firstName;
            this.LastName = lastName;
        }
```

```
        public Person(string firstName, string lastName, int age)
              : this(firstName, lastName)
        {
            this.Age = age;
        }

        public Person(string firstName, string lastName, int age, double salary)
            : this(firstName, lastName, age)
        {
            this.Salary = salary;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var lines = int.Parse(Console.ReadLine());
            var persons = new List<Person>();
            for (int i = 0; i < lines; i++)
            {
                var cmdArgs = Console.ReadLine().Split();
                var person = new Person
                (
                    cmdArgs[0],
                    cmdArgs[1],
                    int.Parse(cmdArgs[2]),
                    double.Parse(cmdArgs[3])
                );
                persons.Add(person);
            }

            var bonus = double.Parse(Console.ReadLine());
            persons.ForEach(person => person.IncreaseSalary(bonus));

            persons.ForEach(p => Console.WriteLine(p.ToString()));
        }
    }
}
```

Задача 3.6. Валидация на данните на класа Box

Всеки от ръбовете на правоъгълния паралелепипед трябва да е неотрицателно число. Разширете класа от предишната задача чрез добавяне на проверка на данните за всеки параметър, даден на конструктора. Направете частен setter, който извършва проверка на данните вътрешно.

Примери

Вход	Изход
2 -3	3 Width cannot be zero or negative.

4	

Решение

```
namespace BoxDataValidation
{
    class Box
    {
        private float lenght;

        public float Lenght
        {
            get { return lenght; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Lenght cannot be zero or
negative.");
                }
                lenght = value;
            }
        }

        private float width;

        public float Width
        {
            get { return width; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Width cannot be zero or
negative.");
                }
                width = value;
            }
        }

        private float height;

        public float Height
        {
            get { return height; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Height cannot be zero or
negative.");
                }
                height = value;
            }
        }
```

```
        public Box(float lenght, float width, float height)
        {
            this.Lenght = lenght;
            this.Width = width;
            this.Height = height;
        }

        public float SurfaceArea()
        {
            return 2 * this.lenght * this.width +
                   2 * this.lenght * this.height +
                   2 * this.width * this.height;
        }

        public float LateralSurfaceArea()
        {
            return 2 * this.lenght * this.height +
                   2 * this.width * this.height;
        }

        public float Volume()
        {
            return this.lenght * this.width * this.height;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            float lenght = float.Parse(Console.ReadLine());
            float width = float.Parse(Console.ReadLine());
            float height = float.Parse(Console.ReadLine());

            Box box = new Box(lenght, width, height);

            Console.WriteLine("Surface Area - {0:f2}", box.SurfaceArea());
            Console.WriteLine("Lateral Surface Area - {0:f2}",
box.LateralSurfaceArea());
            Console.WriteLine("Volume - {0:f2}", box.Volume());
        }
    }
}
```

Задача 3.7. На пазар

Създайте два класа: клас Person и клас Product. Всеки човек трябва да има име, пари и една торба с продукти. Всеки продукт трябва да има име и стойност. Името не може да бъде празен низ. Парите не може да бъдат отрицателно число.

Създайте програма, в която всяка команда отговаря на закупуване на продукт от един обект Person (Човек). Ако човек може да си позволи продукт

го добавя към чантата си. Ако човек не разполага с достатъчно пари, изведете подходящо съобщение (("[Person name] can't afford [Product name]").

На първите два реда са дадени всички хора и всички продукти. След всички покупки да се изведат за всеки човек по реда на въвеждане всички продукти, които той е купил, също в реда на въвеждане на покупките. Ако нищо не е купил, да се изведе името на човека, последвано от "Nothing bought".

При въвеждане на невалидни (отрицателна сума пари да се създаде изключение със съобщение: "Money cannot be negative") или празно име (празно име с изключение със съобщение : "Name cannot be empty") за край на програмата с подходящо съобщение. Вижте примерите по-долу:

Примери

Вход	Изход
Pesho=11;Gosho=4 Bread=10;Milk=2; Pesho Bread Gosho Milk Gosho Milk Pesho Milk END	Pesho bought Bread Gosho bought Milk Gosho bought Milk Pesho can't afford Milk Pesho - Bread Gosho - Milk, Milk
Mimi=0 Kafence=2 Mimi Kafence END	Mimi can't afford Kafence Mimi – Nothing bought
Jeko=-3 Chushki=1; Jeko Chushki END	Money cannot be negative

Решение

```
namespace Shoping
{
    class Product
    {
        private string name;

        public string Name
        {
            get { return name; }
            set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentException("Name cannot be empty");
                }
```

```
            name = value;
        }
    }

    private float price;

    public float Price
    {
        get { return price; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentException("Price cannot be negative");
            }
            price = value;
        }
    }

    public Product(string name, float price)
    {
        this.Name = name;
        this.Price = price;
    }
}

class Person
{
    private string name;

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

    private float money;

    public float Money
    {
        get { return money; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentException("Money cannot be negative");
            }
            money = value;
```

```
            }
        }

        private List<Product> bag;

        public List<Product> Bag
        {
            get { return bag.AsReadOnly().ToList(); }
        }

        public Person(string name, float money)
        {
            this.bag = new List<Product>();
            this.Name = name;
            this.Money = money;
        }

        public void AddProduct(Product item)
        {
            if (this.Money < item.Price)
            {
                throw new ArgumentException($"{this.Name} can't afford
{item.Name}");
            }
            this.Money -= item.Price;
            this.bag.Add(item);
            Console.WriteLine($"{this.Name} bought {item.Name}");
        }

        public override string ToString()
        {
            if (this.Bag.Count == 0) return this.Name + " - Nothing bought";
            else return this.Name + " - " + string.Join(", ", this.bag.Select(x =>
x.Name));
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var persons = new List<Person>();
            var cmd = Console.ReadLine().Split(';');
            foreach (var item in cmd)
            {
                var next = item.Split('=');
                persons.Add
                (
                    new Person(next[0], float.Parse(next[1]))
                );
            }

            var products = new List<Product>();
            cmd = Console.ReadLine().Split(';');
            foreach (var item in cmd)
```

```
            {
                var next = item.Split('=');
                products.Add
                (
                    new Product(next[0], float.Parse(next[1]))
                );
            }

            do
            {
                cmd = Console.ReadLine().Split();
                if (cmd[0] != "END")
                {
                    var product = products.FirstOrDefault(x => x.Name == cmd[1]);
                    try
                    {
                        persons.FirstOrDefault(x => x.Name ==
cmd[0]).AddProduct(product);
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(ex.Message);
                    }
                }
            }
            while (cmd[0] != "END");

            foreach (var person in persons)
            {
                Console.WriteLine(person.ToString());
            }
        }
    }
}
```

Задача 3.8. Първи и резервен Отбор

Създайте клас Team. Добавете към него всички обекти person, които въвеждате. Обектите person, по-млади от 40 отиват в първи отбор, другите - в резервен. На края изведете броя на участниците в първия и резервния отбор.

Класът трябва да има private полета за:

- name: string
- firstTeam: List<Person>
- reserveTeam: List<Person>

Класът трябва да има constructors:

- Team(string name)

Трябва да има също public методи за:

- AddPlayer(Person person): void

- FirstTeam: IReadOnlyCollection
- ReserveTeam: IReadOnlyCollection

Трябва да можете да използвате класа:

```
private string name;
private List<Person> firstTeam;
private List<Person> reserveTeam;

public Team(string name)
{
    this.name = name;
    this.firstTeam = new List<Person>();
    this.reserveTeam = new List<Person>();
}
```

Не бива да използвате клас като този:

```
Team team = new Team("Gorno Nanadolnishte");
foreach (var player in persons)
{
    if (player.Age < 40)
        team.FirstTeam.Add(player);
    else
        team.ReserveTeam.Add(player);
}
```

Примери

Вход	Изход
5 Asen Ivanov 20 2200 Boiko Borisov 57 3333 Ventsislav Ivanov 27 600 Grigor Dimitrov 25 666.66 Boiko Angelov 35 555	First team have 4 players Reserve team have 1 players

Решение

```
namespace Teams
{
    class Person
    {
        private string firstName;

        public string FirstName
        {
            get { return firstName; }
            set { firstName = value; }
        }

        private string lastName;
```

```
        public string LastName
        {
            get { return lastName; }
            set { lastName = value; }
        }

        private int age;

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

        private double salary;

        public double Salary
        {
            get { return salary; }
            set { salary = value; }
        }

        public override string ToString()
        {
            return $"{this.FirstName} {this.LastName} get {this.salary:f2} leva";
        }

        public Person(string firstName, string lastName)
        {
            this.firstName = firstName;
            this.lastName = lastName;
        }

        public Person(string firstName, string lastName, int age)
              : this(firstName, lastName)
        {
            this.age = age;
        }

        public Person(string firstName, string lastName, int age, double salary)
            : this(firstName, lastName, age)
        {
            this.salary = salary;
        }
    }

    class Team
    {
        private string name;
        private List<Person> firstTeam;
        private List<Person> reserveTeam;

        public Team(string name)
        {
```

```
            this.name = name;
            this.firstTeam = new List<Person>();
            this.reserveTeam = new List<Person>();
        }

        public IReadOnlyCollection<Person> FirstTeam
        {
            get { return this.firstTeam.AsReadOnly(); }
        }

        public IReadOnlyCollection<Person> ReserveTeam
        {
            get { return this.reserveTeam.AsReadOnly(); }
        }

        public void AddPlayer(Person player)
        {
            if (player.Age < 40)
            {
                firstTeam.Add(player);
            }
            else
            {
                reserveTeam.Add(player);
            }
        }

        public override string ToString()
        {
            return "First team have " + this.FirstTeam.Count + " players\n"
                 + "Reserve team have " + this.ReserveTeam.Count + " players";
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Team team = new Team("Arsenal");

            int n = int.Parse(Console.ReadLine());
            while (n > 0)
            {
                var line = Console.ReadLine().Split().ToArray();
                team.AddPlayer
                (
                    new Person
                    (
                        line[0],
                        line[1],
                        int.Parse(line[2]),
                        float.Parse(line[3])
                    )
                );
                n--;
```

```
            }

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

Задача 3.9. Създаване на футболен отбор

Футболен отбор има променлив брой играчи, име и рейтинг. Един играч има име и статистика, които са в основата на неговото ниво на умения. Един играч има статистика са издръжливост, Спринт, дрибъл, подавания и стрелба. Всяка статистика може да бъде в диапазона [0..100]. Общото ниво на умение на играч се изчислява като средна стойност на статистиките си. Само името на играча и неговата статистика трябва да бъдат видими за всички от външния свят. Всичко останали данни трябва да бъдат скрити. Отборът трябва да показва име, рейтинг (изчислена от нивата на средните умения на всички играчи в отбора и закръглена до цяло число) и методи за добавяне и премахване на играчи. Вашата задача е да моделирате екипа и играчите, чрез правилното използване на принципите на капсулиране. Покажи само свойствата, които трябва да бъдат видими и валидирайте данните по подходящ начин.

Валидация на данните

- Името не трябва да е празно, null, empty или да е само от интервали. Иначе, изведете "A name should not be empty. "
- Stats трябва да е в обхвата 0..100. Иначе изведете "[Име на статистиката] should be between 0 and 100. "
- Ако получите команда да премахнете липсващ играч, изведете "Player [име на играча] is not in [Име на отбора] team. "
- Ако получите команда да добавите играч към липсващ отбор, изведете "Team [име на отбор] does not exists."
- Ако получите команда за показване на stats за липсващ отбор, изведете "Team [име на отбор] does not exists."

Примери

Вход	Изход
Team;Arsenal Add;Arsenal;Kieran_Gibbs;75;85;84;92;67 Add;Arsenal;Aaron_Ramsey;95;82;82;89;68 Remove;Arsenal;Aaron_Ramsey Rating;Arsenal	Arsenal - 81

END	
Team;Arsenal Add;Arsenal;Kieran_Gibbs;75;85;84;92;67 Add;Arsenal;Aaron_Ramsey;195;82;82;89;68 Remove;Arsenal;Aaron_Ramsey Rating;Arsenal END	Endurance should be between 0 and 100. Player Aaron_Ramsey is not in Arsenal team. Arsenal - 81
Team;Arsenal Rating;Arsenal END	Arsenal – 0

Решение

```
namespace FootballTeam
{
    class Player
    {
        public Player(string name, int du, int sp, int dr, int p, int sh)
        {
            this.Name = name;
            this.Durablility = du;
            this.Sprint = sp;
            this.Dribble = dr;
            this.Passing = p;
            this.Shooting = sh;
        }

        private string name;
        public string Name
        {
            get { return name; }
            set
            {

                if (value.Length == 0 || String.IsNullOrEmpty(value) ||
String.IsNullOrWhiteSpace(value))
                {
                    Console.WriteLine("A name should not be empty");
                }
                else
                {
                    name = value;
                }
            }
        }

        public double Rating()
        {
            return (durability + sprint + dribble + passing + shooting) / 5.0;
```

```
}

private int durability;
private int sprint;
private int dribble;
private int passing;
private int shooting;

public int Durablility
{
    get { return durability; }
    set
    {
        if (value > 100 || value < 0)
        {
            Console.WriteLine("Durability should be between 0 and 100");
        }
        else
        {
            durability = value;
        }
    }
}
public int Sprint
{
    get { return sprint; }
    set
    {
        if (value > 100 || value < 0)
        {
            Console.WriteLine("Sprint should be between 0 and 100");
        }
        else
        {
            sprint = value;
        }
    }
}

public int Dribble
{
    get { return dribble; }
    set
    {
        if (value > 100 || value < 0)
        {
            Console.WriteLine("Dribble should be between 0 and 100");
        }
        else
        {
            dribble = value;
        }
    }
}
```

```
        public int Passing
        {
            get { return passing; }
            set
            {
                if (value > 100 || value < 0)
                {
                    Console.WriteLine("Passing should be between 0 and 100");
                }
                else
                {
                    passing = value;
                }
            }
        }

        public int Shooting
        {
            get { return shooting; }
            set
            {
                if (value > 100 || value < 0)
                {
                    Console.WriteLine("Shooting should be between 0 and 100");
                }
                else
                {
                    shooting = value;
                }
            }
        }
    }
    class Team
    {
        public Team(string name)
        {
            this.Name = name;
            this.Players = new List<Player>();
        }

        private List<Player> players;
        public List<Player> Players
        {
            get { return players; }
            set { players = value; }
        }

        private string name;
        public string Name
        {
            get { return name; }
            set
            {
                if (value.Length == 0 || String.IsNullOrEmpty(value) ||
String.IsNullOrWhiteSpace(value))
```

```
            {
                Console.WriteLine("A name should not be empty");
            }
            else
            {
                name = value;
            }
        }
    }

    public int Rating()
    {
        double sum = this.Players.Sum(x => x.Rating());
        int count = this.Players.Count();
        return (int)Math.Ceiling(sum / count);
    }
}

class Program
{
    static void Main(string[] args)
    {
        List<Team> team = new List<Team>();

        string line = String.Empty;
        while (true)
        {
            line = Console.ReadLine();
            if (line == "END") break;
            var cmd = line.Split(';');
            switch (cmd[0])
            {
                case "Team":
                    {
                        team.Add(new Team(cmd[1]));
                        break;
                    }
                case "Add":
                    {
                        var tm = team.Where(x => x.Name == cmd[1]).First();
                        tm.Players.Add(new Player
                        (
                             cmd[2],
                             int.Parse(cmd[3]),
                             int.Parse(cmd[4]),
                             int.Parse(cmd[5]),
                             int.Parse(cmd[6]),
                             int.Parse(cmd[7])
                        ));
                        break;
                    }
                case "Remove":
                    {
                        var tm = team.Where(x => x.Name == cmd[1]).First();
```

```
                            var pl = tm.Players.Where(x => x.Name ==
cmd[2]).First();
                            tm.Players.Remove(pl);
                            break;
                        }
                    case "Rating":
                        {
                            var tm = team.Where(x => x.Name ==
cmd[1]).Take(1).First();
                            Console.WriteLine("{0} - {1}", tm.Name, tm.Rating());
                            break;
                        }
                }
            }
        }
    }
}
```

Задача 3.10. Калории на Pizza

Пицата е изработена от тесто и различни гарнитури. Вие трябва да създадете класа Pizza, който трябва да има име, тесто и гарнитурата като полета. Всеки вид на съставка трябва да има свой собствен клас. Всяка съставка има различни свойства: тестото може да бъде бяло или пълнозърнесто и освен това то може да бъде хрупкави, chewy или домашно приготвени. Гарнитурата може да бъде от тип месо, зеленчуци, сирене или сос. Всяка съставка трябва да има тегло в грамове и метод за изчисляване на калориите му, според типа си. Калории на грам се изчисляват чрез модификатори. Всяка съставка има 2 калории на грам като база и Модификатор, която дава точна калории. Например, бяло тесто има Модификатор на 1.5, chewy тестото има Модификатор на 1.1, което означава, че бял chewy тесто 100 грама ще има 100 * 1,5 * 1.1 = 330,00 общо калории.

Вашата работа е да създадете класовете по такъв начин, че те правилно да са капсуловани и да предоставят публичен метод за всяка пица която изчислява калориите в зависимост от съставките си.

Step 1. Създайте клас Dough

Основната съставка на пицата е тестото. Първо трябва да създадете един клас за него. Той има тип брашно , който може да бъде бял или пълнозърнест. В допълнение има техника на печене, която може да бъде за хрупкави, жилави или домашно приготвени теста. Тестото трябва да има тегло в грамове. Калории на грам от тестото се изчисляват според типа на брашното и техниката на втастване. Всеки вид тестото има 2 калории на грам като база и Модификатор, който дава точните калории. Например, бялото тесто има Модификатор 1.5, жилавото тесто има Модификатор 1.1, което означава, че бяло жилаво тесто с тегло 100 грама

ще има (2 * 100) * 1,5 * 1.1 = 330,00 общо калории. По-долу Ви се предоставят модификатори:

- White – 1.5;
- Wholegrain – 1.0;
- Crispy – 0.9;
- Chewy – 1.1;
- Homemade – 1.0;

Всичко, което трябва да е видимо за класа е getter за калории на грам. Вашата задача е да се създаде клас с подходящ конструктор, полета, getters и setters. Проверете дали използвате правилните модификатори за достъп.

Step 2. Валидирайте данните за класа Dough

Променете вътрешната логика на класа Dough class чрез добавяне на валидация във setters.

Подсигурете при невалиден вход на типа брашно flour type или невалидна техника на печене подходящо изключение да се връща със съобщение "Invalid type of dough.".

Допустимото тегло на тестото е в диапазона [1..200] грама. Ако е извън диапазона да се върне изключение със съобщение "Dough weight should be in the range [1..200].".

Съобщения на изключенията

- "Invalid type of dough."
- "Dough weight should be in the range [1..200]."

Направете тест на метода main, който въвежда различни видове теста и извежда техните калории, докато се въведе команда "Край".

Примери

Вход	Изход
Dough White Chewy 100 END	330.00
Dough Tip500 Chewy 100 END	Invalid type of dough.
Dough White Chewy 240 END	Dough weight should be in the range [1..200].

Step 3. Създайте клас Topping

След това трябва да създаде класа Topping. Той може да бъде четири различни вида – месо, зеленчуци, сирене или сос. Гарнитурата има тегло в грамове. Калориите на грам гарнитура се изчисляват в зависимост от типа ѝ. Базовите калории на грам са 2. Всеки различен вид гарнитура има модификатор. Например месото има Модификатор 1.5, така че месната гарнитура ще има 1.5 калории на грам (1 * 1.5). Всичко, което трябва да изложи класа е getter за калории на грам. По-долу ви се предоставени модификаторите:

- Meat – 1.2;
- Veggies – 0.8;
- Cheese – 1.1;
- Sauce – 0.9;

Вашата задача е да се създаде клас с подходящ конструктор, полета, getters и setters. Проверете дали използвате правилни модификатори за достъп.

Step 4. Валидиране на данните за класа Topping

Сменете вътрешната логика на класа Topping, като добавите валидация на данните в setter-а.

Уверете се, че гарнитурата е измежду предоставените типове, в противен случай изведете подходящо изключение със съобщение "Cannot place [name of invalid argument] on top of your pizza".

Теглото на гарнитурите е в диапазона [1..50] грама. Ако е извън този диапазон да се върне изключение със съобщението "[Topping type name] weight should be in the range [1..50].".

Съобщения на изключенията

- "Cannot place [name of invalid argument] on top of your pizza."
- "[Topping type name] weight should be in the range [1..50]."

Направете тест на метода main, който въвежда количество тесто и гарнитури и след това извежда техните калории.

Примери

Вход	Изход
Dough White Chewy 100 Topping meat 30 END	330.00 72.00

Dough White chewy 100 Topping Krenvirshi 500 END	330.00 Cannot place Krenvirshi on top of your pizza.
Dough White Chewy 100 Topping Meat 500 END	330.00 Meat weight should be in the range [1..50].

Step 5. Създайте клас Pizza !.

Пицата трябва да има име, няколко гарнитурата и тесто. Използвайте двата класа, които сте направили по-рано. Пицата трябва да имат публични getters за нейното име, брой гарнитури и общото количество калории. Общото количество калории се изчисляват чрез сумиране на калориите на всички съставки, които пицата има. Създайте клас, използвайки подходящ конструктор, направете метод за добавяне на гарнитура, публични getters за тестото и за общото количество калории. Входът за пицата се състои от няколко реда. На първия ред е името на пица и броя на гарнитурите, които има. На втория ред да се въвежда тестото. На следващите редове ще получите всяка гарнитура на пицата. Броят на редовете за гарнитурите се въвежда на първия ред. Ако създаването на пица е успешно да се изведе на един ред името на пицата и общото количество калории в нея.

Step 6. Валидиране на данни за клас Pizza

Името на pizza не трябва да е празен низ. Също не трябва да е повече от 15 символа. Ако не отговаря на това условие, се връща изключение със съобщение "Pizza name should be between 1 and 15 symbols."

Броят на гарнитурите трябва да е в диапазона [0...10]. иначе се връща изключение със съобщение "Number of toppings should be in range [0..10]."

Вашата задача е да изведете името на пицата и общото количество калории в нея, според примера по-долу:

Примери

Вход	Изход
Pizza Meatless 2 Dough Wholegrain Crispy 100 Topping Veggies 50 Topping Cheese 50 END	Meatless – 370.00 Calories.
Pizza Meatfull 5 Dough White cheWy 200 Topping Meat 50	Meatfull – 1028.00 Calories.

Topping Cheese 50 Topping meat 20 Topping sauce 10 Topping Meat 30 END	
Pizza Bulgarian 20 Dough Tip500 Balgarsko 100 Topping Sirene 50 Topping Cheese 50 Topping Krenvirsh 20 Topping Meat 10 END	Number of toppings should be in range [0..10].
Pizza Bulgarian 2 Dough Tip500 Balgarsko 100 Topping Sirene 50 Topping Cheese 50 Topping Krenvirsh 20 Topping Meat 10 END	Invalid type of dough.
Pizza Bulgarian 2 Dough White Chewy 100 Topping Sirene 50 Topping Cheese 50 Topping Krenvirsh 20 Topping Meat 10 END	Cannot place Sirene on top of your pizza.

Решение

Тема 4. Статични полета и методи

Задача 4.1. Дефиниране на клас Person

Създайте клас **Person**. Класът трябва да има:

- name: String - поле
- age: int - поле
- `Name: String - свойство`
- `Age: int – свойство`

Създайте статичен брояч (като поле), който съхранява колко обекта от класа са създадени до момента. Създайте и статично свойство, което да има само get, който да се използва от Main

Решение

```
namespace PersonCounter
{
    class Person
    {
        private string name;

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

        private int age;

        public int Age
        {
            get { return age; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Age must be positive value.");
                }
                age = value;
            }
        }

        public Person(string name, int age)
        {
            this.Name = name;
            this.Age = age;
            counter = counter + 1;
        }

        private static int counter = 0;

        public static int Count()
        {
            return counter;
        }
    }

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

```
            Console.WriteLine("People = {0}", Person.Count());
            Person Ivan = new Person("Ivan", 12);
            Console.WriteLine("People = {0}", Person.Count());
            Person Peter = new Person("Peter", 18);
            Console.WriteLine("People = {0}", Person.Count());
        }
    }
}
```

Задача 4.2. Дефиниране на клас Geometry

Създайте статичен клас **Geometry**. Класът трябва да има следните методи:

- double SquarePerimeter(double side);
- double SquareArea(double side);
- double RectanglePerimeter(double a, double);
- double RectangleArea(double a, double b);
- double CircleArea(double r);

Използвайте всеки един от тях в Main()

Решение

```
namespace GeometryClass
{
    class Geometry
    {
        public static double SquarePerimeter(double side)
        {
            return side * 4.0;
        }

        public static double SquareArea(double side)
        {
            return side * side;
        }

        public static double RectanglePerimeter(double a, double b)
        {
            return 2 * (a + b);
        }

        public static double RectangleArea(double a, double b)
        {
            return a * b;
        }

        public static double CircleArea(double r)
        {
            return Math.PI * Math.Pow(r, 2);
        }
    }

    class Program
    {
```

```
        static void Main(string[] args)
        {
            Console.Write("Square side: ");
            var side = double.Parse(Console.ReadLine());
            var p = Geometry.SquarePerimeter(side);
            var s = Geometry.SquareArea(side);
            Console.WriteLine("Square Perimeter = {0}", p);
            Console.WriteLine("Square Area = {0}", s);

            Console.Write("Rectangle sides [a,b]: ");
            var sides = Console.ReadLine().Split().Select(double.Parse).ToArray();
            p = Geometry.RectanglePerimeter(sides[0], sides[1]);
            s = Geometry.RectangleArea(sides[0], sides[1]);
            Console.WriteLine("Rectangle Perimeter = {0}", p);
            Console.WriteLine("Rectangle Area = {0}", s);

            Console.Write("Circle Radius: ");
            var r = double.Parse(Console.ReadLine());
            s = Geometry.CircleArea(r);
            Console.WriteLine("Circle Area = {0}", s);
        }
    }
}
```

Задача 4.3. Заявка за корен

Напишете клас, който съдържа метод, който връща корен квадратен при подадена заявка. Възможно е да получите голям брой заявки, така че трябва да отговаряте бързо на всяка една от тези заявки.

Реализирайте Main() метод, който да приема едно число – брой на последващите редове. От всеки следващ ред се задава едно цяло число в интервала [1; 1000].

Вход	Изход
5 25 8 3 100 4	5 2.82842712474619 1.73205080756888 10 2

Решение

```
namespace SquareRoot
{
    class SquareRootPrecalculator
    {
        public const int MaxValue = 1000;

        private static double[] sqrtValues;

        static SquareRootPrecalculator()
        {
            sqrtValues = new double[MaxValue + 1];
```

```
            for (int i = 1; i <= MaxValue; i++)
            {
                sqrtValues[i] = Math.Sqrt(i);
            }
        }

        public static double GetSqrt(int number)
        {
            return sqrtValues[number];
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            int n = int.Parse(Console.ReadLine());
            for (int i = 0; i < n; i++)
            {
                var number = int.Parse(Console.ReadLine());
                var value = SquareRootPrecalculator.GetSqrt(number);
                Console.WriteLine(value);
            }
        }
    }
}
```

Задача 4.4. Един магазин

Създайте клас Product с полета за име на продукта и баркод – и двете са текстови низове, цена – double и количество - double. Създайте статичен клас, който да поддържа информация за продуктите в магазина и следните функционалности:

- Продажба на продукт – приема за параметри баркода и продаваното количество. Не допускайте продажба на продукта, ако той има по-малка наличност от желаното количество. Изведете подходящо съобщение на екрана (Sell).
    - Командата ще има вида: Sell <код> <количество>
    - Ако продуктът не съществува или няма достатъчно количество, изведете „Not enough quantity".
- Добавяне на нов продукт – добавя се информация за продукта; баркод, име, цена и количество (Add)
    - Командата ще има вида: Add <код> <име> <количество>
- Зареждане на продукт – добавя се количество от даден продукт; параметрите са баркода и самото количество; не допускайте зареждане на продукт, ако той изобщо не съществува към момента (Update)
    - Командата ще има вида: Update <код> <количество>
    - Ако такъв продукт не съществува, изведете „Please add your product first!"
- Отпечатване на налични продукти по азбучен ред (PrintA)

- Отпечатване на информация за неналични продукти по азбучен ред (PrintU)
- Отпечатване на всички продукти по намаляща наличност - тези от които има най-много са в началото (PrintD)
- Изчисляване на стойността на всички налични продукти (Calculate)

За всичко това трябва да се създаде и програма, която приема команди и изпълнява съответните действия. Името на всяка команда е записано в скоби по-горе. Командата, която приключва въвеждането е „Close". Когато се въведе тя програмата приключва. Всички реални числа се извеждат закръглени и с точно 2 знака след запетаята.

Примери

Вход	Изход	Коментар
Add 359293 ProductA 3.50 8.0 PrintA Sell 359293 8.0 PrintA Update 359293 5.0 PrintA Add 555 ProductB 5.50 3.0 PrintA Sell 359293 4.5 PrintD Calculate Close	ProductA (359293) ProductA (359293) ProductA (359293) ProductB (555) ProductB (555) ProductA (359293) 18.25	• Първата PrintA идва след като само сме добавили продукта, затова на нея съответства само първия ред от изхода. • После продуктът бива продаден и викаме отново PrintA. Този път няма налични продукти и не трябва да отпечатва нищо. • След това зареждаме ново количество и следващото PrintA го отпечатва отново. • Добавяме нов продукт и викаме PrintD: при него първо излиза ProductB, понеже е с по-голямо количество от ProductA, а условието за командата PrintD е продуктите да се извеждат в намаляващ ред, спрямо количеството. • При извикването на Calculate показваме сумата от всички налични продукти: (0.5 * 3.50) + (3*5.50) = 18.25

Решение

```
namespace Shop
{
    class Product
    {
        private string name;

        public string Name
        {
            get { return name; }
            set
            {
```

```
            if (string.IsNullOrEmpty(value))
            {
                throw new ArgumentException("Name must be not empty.");
            }
            name = value;
        }
    }

    private string barcode;

    public string Barcode
    {
        get { return barcode; }
        set
        {
            if (string.IsNullOrEmpty(value))
            {
                throw new ArgumentException("Barcode must not be empty.");
            }
            barcode = value;
        }
    }

    private double price;

    public double Price
    {
        get { return price; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentException("Price must be positive.");
            }
            price = value;
        }
    }

    public Product(string name, string barcode, double price)
    {
        this.Name = name;
        this.Barcode = barcode;
        this.Price = price;
    }
}

class Orders
{
    private static Dictionary<Product, double> orders;

    static Orders()
    {
        orders = new Dictionary<Product, double>();
    }
```

```
        public static void Add(string barcode, string name, double price, double
quantity)
        {
            Product product = new Product(name, barcode, price);
            if (orders.ContainsKey(product))
            {
                orders[product] += quantity;
            }
            else
            {
                orders.Add(product, quantity);
            }
        }

        public static void Sell(string barcode, double quantity)
        {
            if (!orders.ContainsKey(orders.First(item => item.Key.Barcode ==
barcode).Key))
            {
                throw new ArgumentException("Please add your product first");
            }
            var product = orders.First(item => item.Key.Barcode == barcode).Key;

            if (orders[product] < quantity)
            {
                throw new ArgumentException("Not enough quantity");
            }
            orders[product] -= quantity;

        }

        public static void Update(string barcode, double quantity)
        {
            if (!orders.ContainsKey(orders.First(item => item.Key.Barcode ==
barcode).Key))
            {
                throw new ArgumentException("Please add your product first");
            }
            var product = orders.First(item => item.Key.Barcode == barcode).Key;
            orders[product] += quantity;
        }

        public static void PrintA()
        {
            var items = orders.OrderBy(item => item.Key.Name);
            foreach (var item in items)
            {
                Console.WriteLine($"{item.Key.Name} ({item.Key.Barcode})");
            }
        }

        public static void PrintU()
        {
            var items = orders.Where(item => item.Value == 0);
            foreach (var item in items)
```

```
            {
                Console.WriteLine($"{item.Key.Name} ({item.Key.Barcode})");
            }
        }

        public static void PrintD()
        {
            var items = orders.OrderByDescending(item => item.Value);
            foreach (var item in items)
            {
                Console.WriteLine($"{item.Key.Name} ({item.Key.Barcode})");
            }
        }

        public static double Calculate()
        {
            double total = 0;
            foreach (var order in orders)
            {
                total += order.Key.Price * order.Value;
            }
            return total;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var line = Console.ReadLine().Split().ToArray();
            while (line[0] != "Close")
            {
                try
                {
                    switch (line[0])
                    {
                        case "Add":
                            Orders.Add(line[1], line[2], double.Parse(line[3]),
double.Parse(line[4]));
                            break;
                        case "Sell":
                            Orders.Sell(line[1], double.Parse(line[2]));
                            break;
                        case "Update":
                            Orders.Update(line[1], double.Parse(line[2]));
                            break;
                        case "PrintA":
                            Orders.PrintA();
                            break;
                        case "PrintU":
                            Orders.PrintU();
                            break;
                        case "PrintD":
                            Orders.PrintD();
                            break;
```

```
                    case "Calculate":
                        Console.WriteLine("{0:f2}", Orders.Calculate());
                        break;
                }
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
            }

            line = Console.ReadLine().Split().ToArray();
        }
    }
}
```

Задача 4.5. Банкер

Създайте класа **BankAccount**

Този клас трябва да има полета за:

- id: int
- balance: double

Класът трябва да има свойства за:

- ID: int
- Balance: double

Създайте методите:

- **Deposit**(Double amount): void – който да вкарва пари в сметката
- **Withdraw**(Double amount): void – който да изтегля пари от сметката

Заменете метода ToString(), като в този метод изпечатвайте информация за банковата сметка

Създайте статичния клас Bank.

В този клас трябва да създадете следните функционалности:

- Теглене на средства – на този метод трябва да подадете ID-то, списъка с всички създадени сметки и желаната сума за теглене. Ако сумата я няма в наличност или сметката не съществува, изведете подходящо съобщение – за тази функционалност може да се наложи да реализирате няколко метода.
- Внасяне на средства – на този метод трябва да подадете ID-то, списъка с всички създадени сметки и желаната сума за внасяне. Ако сметката не съществува, изведете подходящо съобщение – за тази функционалност може да се наложи да реализирате няколко метода или да използвате вече реализирани такива от предходната точка

Изберете адекватна структура и логика за реализирането на желаните функционалности

Решение

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

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

        private double balance;

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

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

        public void Deposit(double amount)
        {
            this.Balance += amount;
        }

        public void Withdraw(double amount)
        {
            if (amount > this.Balance)
            {
                throw new ArgumentException("Insufficient balance.");
            }
            this.Balance -= amount;
        }

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

```
        }
    }

    class Bank
    {
        private static List<BankAccount> accounts;

        static Bank()
        {
            accounts = new List<BankAccount>();
        }

        public static void Deposit(int id, double amount)
        {
            var account = accounts.Where(item => item.Id == id).FirstOrDefault();
            if (account == null)
            {
                accounts.Add(new BankAccount(id, amount));
            }
            else
            {
                accounts.Where(item => item.Id == id).First().Deposit(amount);
            }
        }

        public static void Withdraw(int id, double amount)
        {
            var account = accounts.Where(item => item.Id == id).FirstOrDefault();
            if (account == null)
            {
                throw new ArgumentException("Bank account does not exists.");
            }
            else
            {
                accounts.Where(item => item.Id == id).First().Withdraw(amount);
            }
        }

        public static void Print()
        {
            accounts.ForEach(account => Console.WriteLine(account));
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var cmd = Console.ReadLine().Split().ToArray();
            while (cmd[0] != "Close")
            {
                try
                {
                    switch (cmd[0])
                    {
```

```
                    case "Deposit":
                        Bank.Deposit(int.Parse(cmd[1]), double.Parse(cmd[2]));
                        break;
                    case "Withdraw":
                        Bank.Withdraw(int.Parse(cmd[1]), double.Parse(cmd[2]));
                        break;
                    case "Print":
                        Bank.Print();
                        break;
                }
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
            }
            cmd = Console.ReadLine().Split().ToArray();
        }
    }
}
```

Тема 5. Допълнителни задачи

Задача 5.1. Превоз на товари

Дадено е транспортно предприятие за превоз на товари. Създайте два класа: клас Truck и клас Freight. Всеки камион трябва да има име, товароносимост и множество товари. Всеки товар трябва да има име и маса (тегло). Името не може да бъде празен низ. Товароносимостта и масата на товарите не може да бъдат отрицателно число.

Създайте програма, в която всяка команда отговаря на товарене на даден камион с даден товар. Ако камионът може да поеме товара - го товари на себе си. Ако камиона не разполага с достатъчна товароносимост, изведете подходящо съобщение (("[Truck] can't loaded [Freight]").

На първите два реда са дадени всички камиони с товароносимости, а на втория всички видове товари с масите си. След всички товарения да се изведат за всеки камион по реда на въвеждане всички товари, с които той е натоварен, също в реда на въвеждане на товарите. Ако нищо не е натоварено на даден камион, да се изведе името на камиона, последвано от "Nothing loaded".

При въвеждане на невалидни (отрицателни маси на товари и товароносимости на камиони да се създаде изключение със съобщение: "Weight cannot be negative") или празно име (празно име с изключение със съобщение : "Name cannot be empty") за край на програмата с подходящо съобщение. Вижте примерите по-долу:

Примери

Вход	Изход
BigTruck=22;Truck=15;LitlleTruck=7; Rock=18;Silk=12;Water=3 BigTruck Rock Truck Water Truck Water Truck Rock LittleTruck Water LittleTruck Water END	BigTruck loaded Rock Truck loaded Water Truck loaded Water Truck loaded Water LittleTruck loaded Water LittleTruck loaded Water BigTruck - Rock Truck – Water, Water, Water LittlleTruck – Water, Water
BigTruck=20;Truck=10;LitlleTruck=7; Rock=18;Silk=12;Water=3;Plastics=2; BigTruck Silk Truck Plastics BigTruck Rock Truck Rock LittleTruck Silk BigTruck Water END	BigTruck loaded Silk Truck loaded Plastics BigTruck can not load Rock Truck can not load Rock LittleTruck can not load Water LittleTruck loaded Water BigTruck loaded Water BigTruck – Silk, Water Truck – Plastics LittlleTruck – Nothing loaded

Решение

```
namespace CarriageOfGoods
{
    class Freight
    {
        private string name;

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

        private double weight;

        public double Weight
        {
```

```
            get { return weight; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Weight cannot be negative");
                }
                weight = value;
            }
        }

        public Freight(string name, double weight)
        {
            this.Name = name;
            this.Weight = weight;
        }

    }

    class Truck
    {
        private string name;

        public string Name
        {
            get { return name; }
            set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentException("Name cannot be empty");
                }
                else name = value;
            }
        }

        private int capacity;

        public int Capacity
        {
            get { return capacity; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("Truck capacity must be positive
value.");
                }
                else capacity = value;
            }
        }

        private List<Freight> freights;

        public IReadOnlyList<Freight> Freights
```

```
        {
            get { return freights.AsReadOnly(); }
        }

        public Truck(string name, int capacity)
        {
            this.freights = new List<Freight>();
            this.Name = name;
            this.Capacity = capacity;
        }

        public void Add(Freight freight)
        {
            if (this.Capacity < freight.Weight)
            {
                throw new ArgumentException($"{this.Name} can't loaded
{freight.Name}");
            }
            else this.freights.Add(freight);
        }

        public override string ToString()
        {
            return this.Name + " - " + string.Join(", ", this.freights.Select(item
=> item.Name));
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var trucks = new List<Truck>();
            var cmd = Console.ReadLine().Split(';');
            foreach (var item in cmd)
            {
                var next = item.Split('=');
                try
                {
                    trucks.Add
                    (
                        new Truck(next[0], int.Parse(next[1]))
                    );
                }
                catch (Exception error)
                {
                    Console.WriteLine(error.Message);
                }
            }

            var freights = new List<Freight>();
            cmd = Console.ReadLine().Split(';');
            foreach (var item in cmd)
            {
                var next = item.Split('=');
```

```
            try
            {
                freights.Add
                (
                    new Freight(next[0], float.Parse(next[1]))
                );
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
            }
        }

        cmd = Console.ReadLine().Split().ToArray();
        while (cmd[0] != "END")
        {
            try
            {
                trucks.FirstOrDefault(item => item.Name == cmd[0]).Add
                (
                    freights.FirstOrDefault(item => item.Name == cmd[1])
                );
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
            }
            cmd = Console.ReadLine().Split().ToArray();
        }

        trucks.ForEach(item => Console.WriteLine(item));
    }
  }
}
```

Задача 5.2. Междузвездни войни

Забележка: Тази задача, както и следваща може да бъдат предложени в различни нюанси – по-прости и по-усложнени. Накратко имаме:

Планети с ресурси – 1 или повече.

Сгради

- Мини , вида им зависи от вида на ресурсите, броят (нивото) им определя скоростта на добив на ресурсите от даден вид
- Корабостроителници, броят (нивото) им определя скоростта на изграждането на кораби

Кораби – няколко различни вида, за изграждането на които се иска наличието на определени ресурси на планетата. Времето за построяването им зависи от броя (нивото) на корабостроителниците на планетата.

Вашата задача е да съставите програма, която

- На 1 ред въвежда Планети с определени запаси от определени ресурси
- На 2 ред въвежда сгради и техните цена, като ресурси, които ще се похарчат за построяването ѝ
- На 3 ред въвежда кораби и техните цени като ресурси, които ще се похарчат за построяването им

Може да се ползва команда TIME [time], която влияе на запасите от ресурсите на планетата, както и на напредъка в корабостроенето. По сценарий – колкото повечее време работят мини и корабостроителници на по-високо ниво (или повече като брой), толкова по бързо добиват единица ресурс или построяват кораб.

TIME [10] – води до увеличаването на ресурсите и корабите пропорционална на 10.

По-долу са дадени два примерни сценария. Ако желаете може да редуцирате ресурси или сгради и мини или кораби.

Създайте три класа: клас Planet, клас Building и клас Ship. Всяка Планета Planet има три полета Name, Metal и Mineral, показващи залежите към текущия момент на планетата. Всяка сграда Building има три полета Name, Metal и Mineral, показващи необходимите ресурси за построяване на дадената сграда. Всеки кораб Ship има три полета Name, Metal и Mineral, показващи необходимите ресурси за построяване на на дадения вид кораб.

Нека имената на сградите са: MetalMine, MineralMine, YardShip, а имената на корабите са: Transporter, Battle Ship, Fighter.

Ако ресурсите са достатъчни за построяването на сградата да се изведе съобщение

„On [Име на планета] was builded [Име на сграда]“

Ако ресурсите са достатъчни за построяването на кораб да се изведе съобщение

„On [Име на планета] was builded [Име на кораб]“

Полето Name на всички обекти не може да е празно, стойностите на Metal и Mineral на Building и Ship са неотрицателни числа. (или поне едно от тях е неотрицателно **) , в противен случай се генерират изключения с подходящи съобщения за възникналите грешки.

Създайте програма, в която всяка команда отговаря на построяване на сграда на дадена планета. Ако Сградата може да се построи, се добавя към

наличните на планетата и се извежда („On [Име а планета] was builded [име на сграда]“)
Ако няма достатъчно ресурси, изведете подходящо съобщение ("[Име на планета] have not resource [Име на ресурс] to build [име на сграда]").

Ако Корабът може да се построи се добавя към наличните на планетата. Ако ресурсите от даден вид не достигат за построяване на сграда, да се генерира изключение със подходящо съобщение („[Име на ресурс] на [Име на Планета] not enough to build a [Име на Сграда]“). Ако ресурсите от даден вид недостигат за построяване на кораба, да се генерира изключение със подходящо съобщение („On [Име на Планета] can not build a [Име на Кораб] [брой] units“). Изведете подходящо съобщение:

- На първия ред се въвеждат всички планети с всички съответни количества ресурси.
- На втори ред се въвеждат всички Сгради с имена и ресурси
- На трети ред се въвеждат всички Кораби с имена и ресурси
- На следващите редове се подават команди за строене на сгради и кораби

След всички построения да се изведат за всяка планета по реда на въвеждане всички сгради и кораби, които са построени, също в реда на въвеждането им.

Ако не са построени сгради, да се изведе "On [име на планета] there are not buildings".

Ако не са построени кораби, да се изведе "On [име на планета] there are not ships".

- При въвеждане на невалидни (отрицателна количества ресурс да се създаде изключение със съобщение: "Amount cannot be negative") или празно име (празно име с изключение със съобщение : "Name cannot be empty")

Команда END – край на въвеждането – извежда на екрана състоянието на планетите:

[Име на планета] [Metal] [количество] [Mineral] [количество]

[Име на планета] MetalMine [нивоl] MineralMine [ниво] ShipYard [ниво]

[Име на планета] Transporter [брой] BatleShip [брой] Fighter [брой]

Забележка: Възможно е да има неточности във форматирането и изходните данни! Да се провери!

Примери

Вход	Изход
Alpha 3000 5000 Beta 4000 2500 Mars 5000 2000 MetalMine 1000 200 MineralMine 100 2000 Transporter 200 300 BattleShip 400 400 Figher 100 100 Alpha MetalMine Alpha MineralMine Beta MetalMine Mars MineralMine Mars MetalMine Beta MineralMine Alpha Transporter 2 Alpha Fihter 1000 Beta BattleShip 4 Alpha MineralMine END	On Alpha was builded MetalMine On Alpha was builded MineralMine On Bet was builded a MetalMine On Mars was builded MineralMine On Mars was builded MetalMine On Beta was builded MineralMine On Alpha was builded Transporter 2 On Alpha can not to build Fighter 1000 units On Beta can not to build BattleShip 4 On Alpha was builded MineralMine Resourses: Alpha Metal 1400 Mineral 200 Beta Metal 2900 Mineral 100 Mars Metal 5000 Mineral 2000 Buildings: Alpha MetalMine 1 MineralMine 2 ShipYard 0 Beta MetalMine 1 MineralMine 1 ShipYard 0 On Mars there are not buildings Ships: Alpha Transporter 2 BatleShip 0 Fighter 0 On Beta there are not ships On Mars there are not ships

Решение

Задача 5.3. Империята отвръща на удара

Добавете към програмата следната функционалност:

- Изминало време – TIME [time] т.е. време, което е изминало и наличните сгради са повлияли на :
    - добива на ресурс по формулата: Ресурс+=брой сгради*100*[time]
    - времето за строенето на кораби, като то намалява при увеличаване на нивото на корабостроителницата (това условие може да се реализира на последващ етап)

Примери
Примерът е както предишния с изключение, че може да се даде команда TIME [време], която по формулата с величини време и ниво на сградите ще повиши ресурсите на планетата.

Ако точно преди END дадем команда TIME 20, за планета Alpha ресурсите ще се повишат.

Вход	Изход
TIME 20	

Решение

Задача 5.4. В най-тъмните подземия

Като млад авантюрист търсиш злато и слава в най-тъмните подземия.

Имате първоначално здраве 100 и първоначални монети 0. Ще ви бъде даден низ, представляващ стаите в подземието, където сте изпратен на мисия. Информацията за всяка стая е разделена от останалите с '|' (вертикална черта): "стая1 | стая2 | стая3 ..."

Всяка стая съдържа име на намерен предмет или чудовище и цяло число, разделени с интервал.

- Ако първата част е "potion", то сте попаднали на отвара, която ви лекува. Увеличете здравето на героя ви с числото във втората част. Но вашето здраве не може да надвишава първоначалното (100). Освен това, ако дадена отвара ви дава възможност да се излекувате над 100, то вие не може да се възползвате напълно от цялата ѝ сила, а само от тази нейна част, която довежда здравето ви до 100. Отпечатайте: "You healed for {0} hp.", където {0} е частта от отварата, от която сте се

възползвали. След това, отпечатайте текущото си здраве: "Current health: {0} hp.".

- Ако първата част е "chest", то вие сте намерили сандък с монети, колкото е числото на втора позиция. Отпечатайте "You found {0} coins." и ги прибавете към вашите.
- Във всеки друг случай сте изправени пред чудовище, ще трябва да се биете. Втората част на информацията на стаята съдържа атаката на чудовището. Трябва да извадите силата на атаката на чудовището от вашето здраве. После:
    - Ако още сте жив (здраве > 0), то вие сте убили чудовището и трябва да се изведе на конзолата "You slayed {monster}."
    - Ако сте умрели, изведете "You died! Killed by {monster}." и най-далечната стая до която сте успели да достигнете: "Best room: {room}". С това вашата мисия приключва.

Ако сте успели да преминете през всички стаи в подземието, изведете на конзолата следващите три реда:

"You've made it!", "Coins: {coins}", "Health: {health}".

Вход

Получавате низ, представляващ стаите в подземието, разделение с '|': "room1|room2|room3...".

Изход

Отпечатайте съответните съобщения, описани по-горе.

Примери

Вход	Изход
rat 10|bat 20|potion 10|rat 10|chest 100|boss 70|chest 1000	You slayed rat. You slayed bat. You healed for 10 hp. Current health: 80 hp. You slayed rat. You found 100 coins. You died! Killed by boss. Best room: 6
Вход	Изход

cat 10|potion 30|orc 10|chest 10|snake 25|chest 110	You slayed cat. You healed for 10 hp. Current health: 100 hp. You slayed orc. You found 10 coins. You slayed snake. You found 110 coins. You've made it! Coins: 120 Health: 65

... игра, в която всеки герой печели деня с блестяща броня и усмивка ...

Решение

Тема 6. Подготовка за изпит

Задача 6.1. Grand Prix

It's racing time again! Welcome to Grand Prix de SoftUni!

Overview

You should write a software which simulates a Formula 1 race under number of commands. Different number of Drivers can participate in each race and each driver ... well, drives a Car. Drivers have different attitude on the track and cars have different specifications which makes the race that thrilling!

Task 1: Structure

Drivers

All drivers have a name, total time record and a car to drive:

Name – a string
TotalTime – a floating-point number
Car - parameter of type Car
FuelConsumptionPerKm – a floating-point number
Speed – a floating-point number

Driver's Speed is calculated throught the formula below. Keep in mind that Speed changes on each lap.
Speed = "(car's Hp + tyre's degradation) / car's fuelAmount"

AggressiveDriver

This type of drivers have FuelConsumptionPerKm equal to 2.7 liters. Also aggressive driver's Speed is multiplied by 1.3.

EnduranceDriver

This type of drivers have FuelConsumptionPerKm equal to 1.5 liters.

Cars

Each car should keep its horsepower (Hp), fuel amount and the type of tyres fit at the moment

- `Hp` – **an integer**
- `FuelAmount` – **a floating-point number**
- `Tyre` - **parameter of type Tyre**

The fuel tank's maximum capacity of each car is 160 liters. Fuel amount cannot become bigger than the tank's maximum capacity. If you are given more fuel than needed you should fill up the tank to the maxiumum and nothing else happens.
If fuel amount drops below 0 liters you should throw an exception and the driver cannot continue the race.

Tyres

Every type of tyre has different hardness of the compound. It also has a degradation level, which is its lifetime:

- `Name` – **a string**
- `Hardness` – **a floating-point number**
- `Degradation` - **a floating-point number**

Every tyre starts with 100 points degradation and drops down towards 0. Upon each lap it's degradation is reduced by the value of the hardness. If a tyre's degradation drops below 0 points the tyre blows up and the driver cannot continue the race. If a tyre blows up you should throw an exeption.

UltrasoftTyre

Because it's ultra-soft this type of tyre has an additional property:

- `Grip` – **a positive floating-point number**

The name of this tyre is always "Ultrasoft".

Upon each lap, it's Degradation drops down by its Hardness summed with its Grip. Also, the ultra-soft tyre blows up when tyre's Degradation drops below 30 points.

HardTyre

The name of this tyre is always "Hard". Hard tyres have less grip and slow down the car but endure bigger distance.

Task 2: Business Logic

Overview: Each execution of the application simulates only one race. In the beginning, you receive info about the track (laps / length) after which drivers are registered. The start of the race is marked by the first CompleteLaps command. The race finishes when all the laps are done by the drivers.

The Controller Class

The business logic of the program should be concentrated around several commands. Implement a class called RaceTower, which will hold the main functionality, represented by these public methods:

RaceTower.cs

```
void SetTrackInfo(int lapsNumber, int trackLength)
{
    //TODO: Add some logic here …
}
void RegisterDriver(List<string> commandArgs)
{
    //TODO: Add some logic here …
}

void DriverBoxes(List<string> commandArgs)
{
    //TODO: Add some logic here …
}

string CompleteLaps(List<string> commandArgs)
{
    //TODO: Add some logic here …
}

string GetLeaderboard()
{
    //TODO: Add some logic here …
}

void ChangeWeather(List<string> commandArgs)
{
    //TODO: Add some logic here …
}
```

NOTE: RaceTower class methods are called from the outside so these methods must NOT receive the command parameter (the first argument from the input line) as part of the arguments!

Method SetTrackInfo() should initialize track's total laps number and length.

Commands

There are several commands that control the business logic of the application and you are supposed to build.

They are stated below.

RegisterDriver Command

Creates a Driver, and registers it into the race. Input data may not be always valid. If you can't create a Driver with the data provided upon this command just skip it. All successfully registered drivers should be saved inside the Racetower class in any type data structure provided by .NET Framework (no custom structures).

Parameters

- `type` – **a string, equal to either "`Aggressive`" or "`Endurance`"**

- `name` – a string
- `hp` – an integer
- `fuelAmount` – a floating-point number
- `tyreType` – a string
- `tyreHardness` – a floating-point number

If the type of tyre is **Ultrasoft**, you will receive 1 extra parameter:

- `grip` – a positive floating-point number

Leaderboard Command

On the first line print:

Lap {current lap}/{total laps number}

On the next lines, all drivers should be displayed in the order of their progress in the following format:

{Position number} {Driver's Name} {Total time / Failure reason}

Drivers are ordered by their **TotalTime** in acsending order.

CompleteLaps Command

Upon this command, all drivers progress the race with the specified number of laps. On each lap, each driver's TotalTime should be increased with the result of the following formula:

"60 / (trackLength / driver's Speed)"

After each lap, you must do the actions below in the exact same order:

1. **Reduce FuelAmount by: "trackLength * driver's fuelConsumptionPerKm".**
2. **Degradate tyre according to its type**

If you are given greater laps number than the number of laps left in the race you should throw an exception with the appropriate message and not increment the completed laps number.

After increasing the TotalTime and decreasing driver's resources (reduce FuelAmount, degradate Tyre) you should check for overtaking opportunities. For more info read the Additional Action -> Overtaking section.

Parameters

- `numberOfLaps` – an integer

Box Command

Makes a driver to box at the current lap which adds 20 seconds to his TotalTime and either changes his tyres with new ones of the specified type or refills with fuel.

Parameters

- **`reasonToBox` - a string, equal to either `ChangeTyres` or `Refuel`**
- **`driversName` - a string specifying which driver boxes**
- **`tyreType` / `fuelAmount` - a string specifying the type of the new tyre / a floating-point specifying how much fuel is refilled**

If the reason is ChangeTyres, you will receive extra parameters:

- **`tyreHardness`- a floating-point specifying the new tyres hardness (only for the ChangeTyres case)**

If the type of tyre is Ultrasoft, you will receive 1 extra parameter:

- **`grip` – a positive floating-point number**

ChangeWeather Command

Changes the current weather. In the beginning, the weather is Sunny by default. Input parameter will always be valid!

Parameters

- **`weather` – a string equal to one of the following: "Rainy", "Foggy", "Sunny"**

Additional Action

DNF

If a driver stops because of some failure he doesn't progress in the race anymore under the CompleteLaps command (his stats get frozen). Drivers that are not racing anymore still take part at the bottom of the Leaderboard in the order of their failure occurrence (the latest failed driver is at the very bottom).

- **The message in case of a blown tyre should be - `"Blown Tyre"`**
- **The message in case of getting out of fuel should be - `"Out of fuel"`**
- **The message in case of a crash should be - `"Crashed"`**

Overtaking

At certain conditions drivers overtake each other. Generally, if a driver is 2 seconds or less behind another driver at the end of a lap, he overtakes the driver ahead which reduces his TotalTime with the same interval of 2 seconds and increases the drivers that has been ahead TotalTime again with the same interval. Although there are some special cases:

- **AggressiveDriver on UltrasoftTyre has an overtake interval up to 3 seconds to the driver ahead and crashes Foggy weather.**
- **EnduranceDriver on HardTyre has an overtake interval up to 3 seconds to the driver ahead and crashes if attempts an overtake in Rainy weather.**

A driver is allowed to attempt an overtake only once in a lap. An overtaken driver is not allowed to fight back for his position in the same lap.

Checking for overtaking opportunities must happen from the slowest (last) driver to the fastest (first).

Task 3: Input / Output

Input

- **On the first line, you will receive an integer representing the number of laps in the race**
- **On the second line, you will receive an integer number representing the length of the track**

- **On the next lines, you will receive different commands. You should stop reading the input when drivers complete all laps in the race**

Below, you can see the format in which each command will be given in the input:

- RegisterDriver {type} {name} {hp} {fuelAmount} {tyreType} {tyreHardness}
- RegisterDriver {type} {name} {hp} {fuelAmount} Ultrasoft {tyreHardness} {grip}
- Leaderboard
- CompleteLaps {numberOfLaps}
- Box Refuel {driversName} {fuelAmount}
- Box ChangeTyres {driversName} Hard {tyreHardness}
- Box ChangeTyres {driversName} Ultrasoft {tyreHardness} {grip}
- ChangeWeather {weather}

Output

Below you can see what output should be provided from the commands.

Leaderboard Command

Lap {current lap}/{total laps number}
On the next lines, all drivers should be displayed in the order of their progress in the following format:

{Position number} {Driver's Name} {Total time / Failure reason}

CompleteLaps Command

If you are given invalid number of laps print on the console:

"There is no time! On lap {current lap}."

In case of a successful overtake you should print on the console:

"{Overtaking driver's name} has overtaken {Overtaken driver's name} on lap {Current lap number}."

Finish

After all laps in the race are completed you should print the winner on the console in the following format:

"{Driver's name} wins the race for {TotalTime} seconds."

The **TotalTime** should be rounded to three digits after the decimal point.

Constraints

- **The `Driver's name` will be a string which may contain any ASCII character, except space (' ').**
- **The `names` of all drivers will always be unique.**

- All drivers will be registered before the race begins (there won't be any RegisterDriver command after the first CompleteLaps command).
- A driver will never box twice in a lap.
- Each race will have a finishing driver.
- There will be NO invalid input data.

Examples

Input	Output
32 3 RegisterDriver Aggressive FirstDriver 650 140 Ultrasoft 0.2 3.8 RegisterDriver Endurance SecondDriver 467 78.48 Hard 0.8 RegisterDriver Endurance ThirdDriver 160 78.48 Ultrasoft 0.4 2.7 CompleteLaps 17 Leaderboard Box Refuel SecondDriver 98.28 CompleteLaps 15	Lap 17/32 1 ThirdDriver 2896.110 2 FirstDriver 6918.938 3 SecondDriver 7209.032 SecondDriver wins the race for 9838.183 seconds.
10 5 RegisterDriver Aggressive FirstDriver 650 140 Ultrasoft 10.2 3.0 RegisterDriver Aggressive SecondDriver 650 140 Hard 3.9 RegisterDriver Endurance ThirdDriver 360 78.48 Ultrasoft 2.4 0.7 CompleteLaps 14 CompleteLaps 8 Leaderboard Box ChangeTyres ThirdDriver Hard 0.3 CompleteLaps 2	There is no time! On lap 0. FirstDriver has overtaken SecondDriver on lap 1. Lap 8/10 1 ThirdDriver 931.587 2 SecondDriver 1127.098 3 FirstDriver Blown Tyre ThirdDriver wins the race for 1752.693 seconds.
14 5 RegisterDriver Endurance FirstDriver 650 140 Hard 0.2 RegisterDriver Endurance SecondDriver 650 140 Ultrasoft 0.2 0.3 RegisterDriver Aggressive ThirdDriver 350 100 Ultrasoft 0.2 0.3 RegisterDriver Aggressive FourthDriver 450 60 Hard 1.2 ChangeWeather Rainy CompleteLaps 1 Leaderboard Box Refuel FourthDriver 168 CompleteLaps 6 Box Refuel FourthDriver 2 CompleteLaps 6 Leaderboard CompleteLaps 1	Lap 1/14 1 SecondDriver 64.286 2 ThirdDriver 70.200 3 FourthDriver 143.000 4 FirstDriver Crashed Lap 13/14 1 SecondDriver 1353.124 2 FourthDriver 2122.949 3 ThirdDriver Out of fuel 4 FirstDriver Crashed SecondDriver wins the race for 1563.054 seconds.

Task 4: Bonus

Factories

You know, that the keyword new is a bottleneck and we are trying to use it as less as possible. We even try to separate it in new classes. These classes are called

Factories and the convention for them is {TypeOfObject}Factory. You need to have two different factories, one for Driver and one for Tyre. This is actually a design pattern and you can read more about it. Factory Pattern

Points

For all tasks, you can submit the same project. Every different task gives you points:

Task 1. 120 points
Task 2. 180 points
Task 3. 100 points
Task 4. 25 points

Задача 6.2. Minedraft

You ever heard about the Rick and Morty's Foundation for mining Plumbus Ore. Naaa, probably not. Well let's just say there is this company that mines things, and they hired you to write them a supervising software. A draft which will be used to analyze the data of the mining – a … Minedraft.

Overview

The main system consists of Harvesters and Providers. The Harvesters are the ones that make real money – they mine Plumbus Ore. But they need a large amount of energy to do that. That's where the Providers come. The Providers are the ones that provide the energy for the harvesters.

Task 1: Structure

The Structure consists of Harvesters and Providers.

Harvesters

A basic Harvester has the following properties:

- **id – a string.**
- **oreOutput – a floating-point number.**
- **energyRequirement – a floating-point number.**

For all harvesters you need to validate, that ore output and energy requirement for each harvester is NOT negative. Also you need to validate that energy requirement for each harveter is NOT over 20000. There *are generally 2 types of Harversters:*

SonicHarvester

Really fast… Has an extra property:

- **`sonicFactor` – an integer.**

UPON INITIALIZATION, divides its given energyRequirement by its sonicFactor.

HammerHarvester

Heavy… and big.

UPON INITIALIZATION, increases its oreOutput by 200 %, and increases its energyRequirement by 100 %.

Providers

A basic **Provider** has the following properties:

- `id` – **a string.**
- `energyOutput` – **a floating-point number.**

Every provider energy output need to be positive numbers, less than 10000. There are generally 2 types of Providers:

SolarProvider

Extracts energy from the Sun. Nothing special here.

PressureProvider

Extracts energy from deep beneath the earth. Temperatures and Pressure affect it.
UPON INITIALIZATION, increases its energyOutput by 50 %.

Task 2: Business Logic

The Controller Class

The business logic of the program should be concentrated around several commands. Implement a class called DraftManager, which will hold the main functionality, represented by these public methods:

DraftManager.cs

```
string RegisterHarvester(List<string> arguments)
{
    //TODO: Add some logic here …
}
string RegisterProvider(List<string> arguments)
{
    //TODO: Add some logic here …
}
string Day()
{
    //TODO: Add some logic here …
}
string Mode(List<string> arguments)
{
    //TODO: Add some logic here …
}
string Check(List<string> arguments)
{
    //TODO: Add some logic here …
}
string ShutDown()
{
    //TODO: Add some logic here …
}
```

NOTE: DraftManager class methods are called from the outside so these methods must NOT receive the command parameter as part of the arguments!

Functionality

The whole system works on 3 modes – "Full Mode", "Half Mode", "Energy Mode". Depending on the mode, the Harvesters and Providers work differently. By DEFAULT the mode is "Full Mode".

The Providers and Harvesters have ids, which will always be unique.

When a day passes, the Providers produce energy and the Harvesters consume energy and mine Plumbus Ore. In your program a day passes when you have been given the corresponding command.

The Providers produce energy which is being stored on the system. When there is ENOUGH energy to power up ALL Harvesters, the Harvesters consume it and return the ore.

The system keeps the totalStoredEnergy and the totalMinedOre.

Modes

The different modes make the Harvesters work differently. You can save more power by changing the modes and stalling them a little. The Providers remain unaffected by the modes.

Full Mode

All Harvesters consume their FULL energy requirements, and produce their FULL ore output.

Half Mode

All Harvesters consume 60 % of their energy requirements, and produce 50 % of their ore output.

Energy Mode

The Harvesters consume nothing, and produce nothing. They practically do NOT work.

Commands

There are several commands that control the business logic of the application you are supposed to build.

They are stated below.

RegisterHarvester Command

Creates a Harvester, and registers it into the system, so they can start mining when new day come.

Parameters

- `type` – **a string, equal to either** `Sonic` **or** `Hammer`**.**
- `id` – **a string.**
- `oreOutput` – **a positive floating-point number.**
- `energyRequirement` – **a positive floating-point number.**

If the type is Sonic, you will receive 1 extra parameter:

- `sonicFactor` – **a positive integer.**

RegisterProvider Command

Creates a Provider, and registers it into the system. They start to provide energy from next day.

Parameters

- `type` – **a string, equal to either** `Solar` **or** `Pressure`**.**
- `id` – **a string.**
- `energyOutput` – **a positive floating-point number.**

Day Command

When you receive this command a day passes. This is the moment where real work starts. You need to calculate all the provided energy and STORE it in the system. Then you need to check if there is enough energy for harvesters to start mining. If the sum of energy requirement of ALL harvesters is more than the energy stored then NOTHING happens. If there is enough energy, ALL harvesters start mining and they consume from the stored energy EQUAL to their energy requirement.

NOTE: The summed up energyRequirement might be less or more depending on the current working Mode.

Mode Command

Changes the mode of the system, to the given one.

Parameters

- `mode` **- a string, equal to either** `Full`**,** `Half` **or** `Energy`**.**

Check Command

Checks the Provider or the Harvester with the given id, returning a string representation of it. The system should check if there is an element with the given id among the Providers or the Harvesters. The ids are unique so there should be only one with that id.

Parameters

- `id` – **a string.**

Shutdown Command

Ends the program and print total energy stored and ore mined

Task 3: Input / Output

Input

Below, you can see the format in which each command will be given in the input:

- RegisterHarvester {type} {id} {oreOutput} {energyRequirement}
- RegisterHarvester Sonic {id} {oreOutput} {energyRequirement} {sonicFactor}
- RegisterProvider {type} {id} {energyOutput}
- Day

- Mode {mode}
- Check {id}
- Shutdown

Output
Below you can see what output should be provided from the commands.

RegisterHarvester Command
Successful command should print "Successfully registered {type} Harvester – {id}".

Unsuccessfull comand: "Harvester is not registered, because of it's {propertyName}"

RegisterProvider Command
Should output a message "Successfully registered {type} Provider – {id}".

Unsuccessfull comand: "Provider is not registered, because of it's {propertyName}"

Day Command
Should output a message

"A day has passed.

"Energy Provided: {summedEnergyOutput}".

"Plumbus Ore Mined: {summedOreOutput}".

The **summedEnergyOutput** and **summedOreOutput** are the ones that have been mined for the day.

Mode Command
Should output a message "Successfully changed working mode to {mode} Mode".

Check Command
Should return a string representation of the element with the given id. If there is no such element, the command should output a message "No element found with id – {id}".

Because the element can either be a Provider or a Harvester, both output formats have been provided below:

Harvester	Provider
"{type} Harvester – {id} Ore Output: {oreOutput} Energy Requirement: {energyRequired}"	"{type} Provider – {id} Energy Output: {energyOutput}"

Shutdown Command

Should output a message

"System Shutdown

Total Energy Stored: {totalEnergyStored}

Total Mined Plumbus Ore: {totalMinedOre}".

The **totalEnergyStored** and **totalMinedOre** are the total values that have been gathered throughout the program's execution.

Constraints

- The `id` will be a string which may contain any ASCII character, except space (' ').
- The `ids` will always be unique.
- All floating-point numbers will be in range [-1.000.000, 1.000.000].
- The `sonicFactor` will be in range [1, 10].
- There will be NO invalid input data.

Examples

Input	Output
RegisterHarvester Sonic AS-51 100 100 10 RegisterHarvester Hammer CDD 100 50 RegisterProvider Solar Falcon 100 Day Check AS-51 Check CDD Check Falcon Day Shutdown	Successfully registered Sonic Harvester - AS-51 Successfully registered Hammer Harvester - CDD Successfully registered Solar Provider - Falcon A day has passed. Energy Provided: 100 Plumbus Ore Mined: 0 Sonic Harvester - AS-51 Ore Output: 100 Energy Requirement: 10 Hammer Harvester - CDD Ore Output: 300 Energy Requirement: 100 Solar Provider - Falcon Energy Output: 100 A day has passed. Energy Provided: 100 Plumbus Ore Mined: 400 System Shutdown Total Energy Stored: 90 Total Mined Plumbus Ore: 400
RegisterHarvester Sonic AS-51 100 1000000 10 RegisterHarvester Hammer CDD 100 50	Harvester is not registered, because of it's EnergyRequirement

RegisterProvider Solar Falcon 100 RegisterProvider Solar Pesho 100000 Day Check CDD Check Falcon Day Shutdown	Successfully registered Hammer Harvester - CDD Successfully registered Solar Provider - Falcon Provider is not registered, because of it's EnergyOutput A day has passed. Energy Provided: 100 Plumbus Ore Mined: 300 Hammer Harvester - CDD Ore Output: 300 Energy Requirement: 100 Solar Provider - Falcon Energy Output: 100 A day has passed. Energy Provided: 100 Plumbus Ore Mined: 300 System Shutdown Total Energy Stored: 0 Total Mined Plumbus Ore: 600
RegisterProvider Pressure Deep-1 1000 RegisterProvider Pressure Deep-3 2000 Day Mode Energy RegisterHarvester Hammer S-1 10000 11250 Day Check Something Check S-1 Mode Half Day Shutdown	Successfully registered Pressure Provider - Deep-1 Successfully registered Pressure Provider - Deep-3 A day has passed. Energy Provided: 4500 Plumbus Ore Mined: 0 Successfully changed working mode to Energy Mode Harvester is not registered, because of it's EnergyRequirement A day has passed. Energy Provided: 4500 Plumbus Ore Mined: 0 No element found with id - Something No element found with id - S-1 Successfully changed working mode to Half Mode A day has passed. Energy Provided: 4500 Plumbus Ore Mined: 0 System Shutdown Total Energy Stored: 13500 Total Mined Plumbus Ore: 0

Task 4: Bonus

Abstraction

Probably, you have already noticed, that there is a way to improve the abstraction of your code. If NOT, now is the time to think about this. For this task, you need one more level of abstraction for Harvester and Providers.

Factories

You know, that the keyword new is a bottleneck and we are trying to use it as less as possible. We even try to separate it in new classes. These classes are called Factories and the convention for them is {TypeOfObject}Factory. You need to have two different factories, one for Harvesters and one for Providers. This is actually a design pattern and you can read more about it. Factory Pattern

Points

For all tasks you can submit same project. Every different task give you points:

Task 5. **100 points**
Task 6. **200 points**
Task 7. **100 points**
Task 8. **50 points**

Задача 6.3. Dungeons and Code Wizards

The time has come for your OOP Basics exam. It will be a perilous journey, but if you manage to keep a cool head, you'll be able to get through it with minimal damage.

Overview

In this exam, you need to build a Dungeons and Dragons-esque project, which has support for **characters**, **items** and **inventories** for storing each character's items. The project will consist of the **entity classes** and a **controller class**, which manages the **interaction** between the characters and items.

Task 1: Structure (150 Points)

Item

This is a **base class** for any **items** and it **should not be able to be instantiated**.

Data

Weight – an **integer number**

Behavior

Each **item** has the following **behavior**:

Void AffectCharacter(Character Character)

For an item to affect a character, the character **needs to be alive**.

If not, **throw** an **InvalidOperationException with the message "Must be alive to perform this action!"**.

Throw this exception everywhere a character needs to be alive to perform the action.

Constructor
An **item** should take the following values upon initialization:

int weight

HealthPotion
The **health potion** always has a **weight** of **5**.
Behavior
Each **HealthPotion** has the following **behavior**:

Void AffectCharacter(Character Character)

For an item to affect a character, the character **needs to be alive**.

The character's **health** gets **increased** by **20 points**.

Constructor
An **item** should be able to be instantiated **without any parameters**.

PoisonPotion
The **poison potion** always has a **weight** of **5**.
Behavior
Each **PoisonPotion** has the following **behavior**:

Void AffectCharacter(Character Character)

For an item to affect a character, the character **needs to be alive**.

The character's **health** gets **decreased** by **20 points**. If the character's health **drops to zero**, the character **dies** (**IsAlive ➔ false**).

Constructor
A **PoisonPotion** should be able to be instantiated **without any parameters**.

ArmorRepairKit
The armor repair kit always has a **weight** of **10**.
Behavior
Each **ArmorRepairKit** has the following **behavior**:

Void AffectCharacter(Character Character)

For an item to affect a character, the character **needs to be alive**.

The character's **armor** restored up to the **base armor** value.

Example: Armor: 10, Base Armor: 100 ➔ Armor: 100

Constructor
An **ArmorRepairKit** should be able to be instantiated **without any parameters**.

Bag
This is a **base class** for any **bags** and it **should not be able to be instantiated**.

Data

Capacity – an **integer number**. Default value: 100
Load – Calculated property. The **sum of the weights** of the **items** in the bag.
Items – **Read-only collection** of type **Item**

Behavior

Each **bag** has the following **behavior**:

Void AddItem(Item Item)

If the current load + the weight of the item attempted to be added is **greater than** the bag's **capacity**, throw an **InvalidOperationException** with the message **"Bag is full!"**

If the check passes, the **item** is added to the **bag**.

Item GetItem(string Name)

If no items exist in the bag, throw an **InvalidOperationException** with the message **"Bag is empty!"**

If an item with that **name doesn't exist** in the bag, throw an **ArgumentException** with the message **"No item with name {name} in bag!"**

If both checks pass, the **item** is removed from the **bag** and **returned** to the **caller**.

Constructor

A **Bag** should take the following values upon initialization:

int capacity

Backpack

This is a **type of bag** with 100 capacity.

Satchel

This is a **type of bag** with 20 capacity.

Character

This is a **base class** for any **characters** and it **should not be able to be instantiated**.

Data

- `Name` – a string (cannot be null or whitespace).
  - Throw an ArgumentException with the message "Name cannot be null or whitespace!"
- BaseHealth – a floating-point number
- `Health` – a floating-point number (current health).
  - Health maxes out at BaseHealth and mins out at 0.
- BaseArmor – a floating-point number
- `Armor` – a floating-point number (current armor)
  - Armor maxes out at BaseArmor and mins out at 0.
- AbilityPoints – a floating-point number

- `Bag` – a parameter of type Bag
- `Faction` – a constant value with 2 possible values: CSharp and Java
- `IsAlive` – boolean value (default value: `True`)
- `RestHealMultiplier` – a floating-point number (default: 0.2), could be overriden

Behavior

Each **character** has the following **behavior**:

Void TakeDamage(double HitPoints)

For a character to take damage, they need to **be alive**.

The character takes damage equal to the **hit points**. Taking damage works like so:

The character's **armor** is **reduced** by the **hit point amount**, then if there are **still hit points left**, they take that amount of **health damage**.

If the character's **health** drops to **zero**, the character **dies** (**IsAlive** become **false**)

Example: Health: **100**, Armor: **30**, Hit Points: **40** ➔ Health: **90**, Armor: **0**

Void Rest()

For a character to rest, they need to **be alive**.

The character's **health** increases by their **BaseHealth**, multiplied by their **RestHealMultiplier**

Example: **Health**: **50**, **BaseHealth**: **100**, **RestHealMultiplier**: **0.2** ➔ **Health**: 50 + (100 * 0.2) ➔ 70

Void UseItem(Item Item)

For a character to use an item, they need to be **alive**.

The item affects the character with the item effect.

Void UseItemOn(Item Item, Character Character)

For a character to use an item on another character, **both of them** need to be **alive**.

The item affects the targeted character with the item effect.

Void GiveCharacterItem(Item Item, Character Character)

For a character to give another character an item, **both of them** need to be **alive**.

The targeted character **receives the item**.

Void ReceiveItem(Item Item)

For a character to receive an item, they need to be **alive**.

The character puts the **item** into their **bag**.

Constructor
A **character** should take the following values upon initialization:

string name, double health, double armor, double abilityPoints, Bag bag, Faction faction

IAttackable
A **contract** for any class that **implements it**, that includes an "**Attack(Character character)**" method

IHealable
A **contract** for any class that **implements it**, that includes a "**Heal(Character character)**" method

Warrior
The **Warrior** is a special class, who can **attack** other characters.

Data
The Warrior class always has **100 Base Health**, **50 Base Armor**, **40 Ability Points**, and a **Satchel** as a bag.

Constructor
The **Warrior** only needs a **name** and a **faction** for initialization:

string name, Faction faction

Behavior
The warrior only has **one special behavior** (every other behavior is **inherited**):

Void Attack(Character Character)

For a character to attack another character, **both of them need to be alive**.

If the character they are trying to attack is the same character, throw an **InvalidOperationException** with the message "**Cannot attack self!**"

If the character the character is attacking is from the **same faction**, throw an **ArgumentException** with the message "**Friendly fire! Both characters are from {faction} faction!**"

If all of those checks pass, the **receiving character takes damage** with **hit points** equal to the **attacking character's ability points**.

Cleric
The **Cleric** is a special class, who can **heal** other characters.

Data
The Cleric class always has **50 Base Health**, **25 Base Armor**, **40 Ability Points**, and a **Backpack** as a bag.

The cleric's **RestHealMultiplier** is **0.5**.

Constructor

The **Cleric** only needs a **name** and a **faction** for initialization:

string name, Faction faction

Behavior

The cleric only has **one special behavior** (every other behavior is **inherited**):

Void Heal(Character Character)

For a character to heal another character, **both of them need to be alive**.

If the character the character is healing is from a **different faction**, throw an **InvalidOperationException** with the message **"Cannot heal enemy character!"**

If both of those checks pass, the **receiving character's health increases by the healer's ability points**.

Task 2: Business Logic (200 Points)

The Controller Class

The business logic of the program should be concentrated around several **commands**. Implement a class called **DungeonMaster**, which will hold the **main functionality**.

The Dungeon Master keeps track of the **character party** and the **item pool** (the items in the game, which can be picked up). It also keeps track of the **last survivor's consecutive rounds (explained below)**.

Note: The DungeonMaster class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!

The main functionality is represented by these **public methods**:

DungeonMaster.cs

```
public string JoinParty(string[] args)
{
    throw new NotImplementedException();
}

public string AddItemToPool(string[] args)
{
    throw new NotImplementedException();
}

public string PickUpItem(string[] args)
{
    throw new NotImplementedException();
}

public string UseItem(string[] args)
{
    throw new NotImplementedException();
}
```

```
public string UseItemOn(string[] args)
{
    throw new NotImplementedException();
}

public string GiveCharacterItem(string[] args)
{
    throw new NotImplementedException();
}

public string GetStats()
{
    throw new NotImplementedException();
}

public string Attack(string[] args)
{
    throw new NotImplementedException();
}

public string Heal(string[] args)
{
    throw new NotImplementedException();
}

public string EndTurn(string[] args)
{
    throw new NotImplementedException();
}

public bool IsGameOver()
{
     throw new NotImplementedException();
}
```

NOTE: DungeonMaster class methods are called from the outside so these methods must NOT receive the command parameter (the first argument from the input line) as part of the arguments!

ALSO NOTE: The **DungeonMaster** class should not handle any exceptions. That should be the responsibility of the class, which reads the commands and passes them to the **DungeonMaster**.

Commands

There are several commands that control the business logic of the application and you are supposed to build.
They are stated below.

JoinParty Command

Parameters

- `faction` – **a string**
- `characterType` – **string**
- `name` – **string**

Functionality

Creates a character and **adds them** to the **party**.

If the **faction** is invalid, throw an **ArgumentException** with the message **"Invalid faction "{faction}"!"**

If the **character type** is invalid, throw an **ArgumentException** with the message **"Invalid character type "{characterType}"!"**

Returns the **string "{name} joined the party!"**

AddItemToPool Command

Parameters

- `itemName` **–string**

Functionality

Creates an item and **adds it** to the **item pool**.

If the **item type** is **invalid**, throw an **ArgumentException** with the message **""Invalid item "{name}"!"**

Returns the **string "{itemName} added to pool."**

PickUpItem Command

Parameters

- `characterName` **– string**

Functionality

Makes the character with the specified name **pick up the last item in the item pool**.

If the character doesn't exist in the **party**, throw an **ArgumentException** with the message **"Character {name} not found!"**

If there's **no items left** in the pool, throw an **InvalidOperationException** with the message **"No items left in pool!"**

Returns the **string "{characterName} picked up {itemName}!"**

UseItem Command

Parameters

- `characterName` **– a string**
- `itemName` **– string**

Functionality

Makes the character with that name use an **item with that name from their bag**.

If the character doesn't exist in the **party**, throw an **ArgumentException** with the message **"Character {name} not found!"**

The rest of the exceptions should be processed by the called functionality (empty bag, etc.)

Returns the **string "{character.Name} used {itemName}."**

UseItemOn Command

Parameters

- `giverName` – **a string**
- `receiverName` – **string**
- `itemName` – **string**

Functionality

Makes the giver get an **item with that name from their bag** and **uses it on the receiving character**.

Process any edge cases (giver not found, receiver not found, item not found, etc.) in the same way as in the above commands.

Returns the **string "{giverName} used {itemName} on {receiverName}."**

GiveCharacterItem Command

Parameters

- `giverName` – **a string**
- `receiverName` – **string**
- `itemName` – **string**

Functionality

Makes the giver get an **item with that name from their bag** and **gives it to the receiving character**.

Process any edge cases (giver not found, receiver not found, item not found, etc.) in the same way as in the above commands.

Returns the **string "`{giverName} gave {receiverName} {itemName}.`"**

GetStats Command

Parameters

No parameters.

Functionality

Returns info about **all characters**, sorted by **whether they are alive** (**descending**), **then by** their **health** (**descending**)

The format of a single character is:

{name} - HP: {health}/{baseHealth}, AP: {armor}/{baseArmor}, Status: {Alive/Dead}
Returns the formatted character info for each character, **separated by new lines**.

Attack Command

Parameters

- `attackerName` – **a string**
- `receiverName` – **string**

Functionality

Makes the **attacker attack** the **receiver**.

If any of the characters don't exist in the party, throw exceptions with messages just like the above commands.

If the **attacker cannot attack**, throw an **ArgumentException** with the message "**{attacker.Name} cannot attack!**"

The command output is in the following format:

{attackerName} attacks {receiverName} for {attacker.AbilityPoints} hit points! {receiverName} has {receiverHealth}/{receiverBaseHealth} HP and {receiverArmor}/{receiverBaseArmor} AP left!
If the attacker ends up **killing** the receiver, add a **new line**, plus "**{receiver.Name} is dead!**" to the output.

Returns the **formatted string**

Heal Command

Parameters

- `healerName` – **a string**
- `healingReceiverName` – **string**

Functionality

Makes the **healer heal** the **healing receiver**.

If any of the characters don't exist in the party, throw exceptions with messages just like the above commands.

If the **healer cannot heal**, throw an **ArgumentException** with the message "**{healerName} cannot heal!**"

The command **output** is in the following format:

{healer.Name} heals {receiver.Name} for {healer.AbilityPoints}! {receiver.Name} has {receiver.Health} health now!

Returns the **formatted string**

EndTurn Command

Parameters

No parameters.

Functionality

Ends the turn. Several things happen when a turn ends.

First, each **alive** character **rests**. Then the line "**{character.Name} rests ({healthBeforeRest} => {currentHealth})**" is added to the output.

If there are **one or zero alive** characters left, the **last survivor rounds** are **incremented by one**.

Returns all the "**x rests...**" commands, separated by new lines.

IsGameOver Command

Parameters

No parameters.

Functionality

If the **last survivor** survives **alone more than one round**, the game is **over**.

The command returns whether the game is over or not (**true/false**)

Task 3: Input / Output (100 Points)

Input

- You will receive commands until the game is over or until the command you read is null or empty.

Below, you can see the format in which each command will be given in the input:

- JoinParty {Java/CSharp} {class} {name}
- AddItemToPool {itemName}
- PickUpItem {characterName}
- UseItem {characterName} {itemName}
- UseItemOn {giverName} {receiverName} {itemName}
- GiveCharacterItem {giverName} {receiverName} {itemName}
- GetStats
- Attack {attackerName} {attackTargetName}
- Heal {healerName} {healingTargetName}
- EndTurn
- IsGameOver

Output

Print the output from each command when issued. When the game is over, print "**Final stats:**" and the output from the **GetStats** command.

If an exception is thrown during any of the commands' execution, print:

- "`Parameter Error: `" plus the message of the exception if it's an `ArgumentException`
- "`Invalid Operation: `" plus the message of the exception if it's an `InvalidOperationException`

Constraints

- The commands will always be in the provided format.

Examples

Input

```
JoinParty CSharp Warrior Gosho
JoinParty Java Warrior Pesho
AddItemToPool HealthPotion
AddItemToPool ArmorRepairKit
AddItemToPool PoisonPotion
PickUpItem Gosho
EndTurn
JoinParty Java Cleric Ivan
Attack Gosho Pesho
Attack Gosho Pesho
EndTurn
Attack Gosho Pesho
Heal Ivan Pesho
EndTurn
Attack Gosho Ivan
Attack Gosho Ivan
Attack Gosho Pesho
Attack Gosho Pesho
Attack Gosho Pesho
EndTurn
EndTurn
```

Output

```
Gosho joined the party!
Pesho joined the party!
HealthPotion added to pool.
ArmorRepairKit added to pool.
PoisonPotion added to pool.
Gosho picked up PoisonPotion!
Gosho rests (100 => 100)
Pesho rests (100 => 100)
Ivan joined the party!
Gosho attacks Pesho for 40 hit points! Pesho has 100/100 HP and 10/50 AP
left!
Gosho attacks Pesho for 40 hit points! Pesho has 70/100 HP and 0/50 AP
left!
Gosho rests (100 => 100)
Pesho rests (70 => 90)
Ivan rests (50 => 50)
Gosho attacks Pesho for 40 hit points! Pesho has 50/100 HP and 0/50 AP
left!
Ivan heals Pesho for 40! Pesho has 90 health now!
Gosho rests (100 => 100)
Pesho rests (90 => 100)
Ivan rests (50 => 50)
Gosho attacks Ivan for 40 hit points! Ivan has 35/50 HP and 0/25 AP left!
```

```
Gosho attacks Ivan for 40 hit points! Ivan has 0/50 HP and 0/25 AP left!
Ivan is dead!
Gosho attacks Pesho for 40 hit points! Pesho has 60/100 HP and 0/50 AP
left!
Gosho attacks Pesho for 40 hit points! Pesho has 20/100 HP and 0/50 AP
left!
Gosho attacks Pesho for 40 hit points! Pesho has 0/100 HP and 0/50 AP
left!
Pesho is dead!
Gosho rests (100 => 100)
Gosho rests (100 => 100)
Final stats:
Gosho - HP: 100/100, AP: 50/50, Status: Alive
Pesho - HP: 0/100, AP: 0/50, Status: Dead
Ivan - HP: 0/50, AP: 0/25, Status: Dead
```

Input
JoinParty CSharp Warrior Gosho JoinParty CSharp Warrior Pesho AddItemToPool HealthPotion AddItemToPool PoisonPotion PickUpItem Pesho PickUpItem Gosho PickUpItem Pesho UseItem Pesho HealthPotion UseItem Pesho PoisonPotion UseItemOn Gosho Pesho HealthPotion AddItemToPool PoisonPotion PickUpItem Gosho GiveCharacterItem Gosho Pesho PoisonPotion JoinParty Java Warrior Ivan Attack Ivan Gosho Attack Ivan Gosho Attack Ivan Gosho Attack Gosho Ivan Attack Ivan Gosho EndTurn Attack Ivan Pesho Attack Ivan Pesho Attack Ivan Pesho Attack Ivan Pesho EndTurn EndTurn
Output

```
Gosho joined the party!
Pesho joined the party!
HealthPotion added to pool.
PoisonPotion added to pool.
Pesho picked up PoisonPotion!
Gosho picked up HealthPotion!
Invalid Operation: No items left in pool!
Invalid Operation: No item with name HealthPotion in bag!
Pesho used PoisonPotion.
Gosho used HealthPotion on Pesho.
PoisonPotion added to pool.
Gosho picked up PoisonPotion!
Gosho gave Pesho PoisonPotion.
Ivan joined the party!
Ivan attacks Gosho for 40 hit points! Gosho has 100/100 HP and 10/50 AP
left!
Ivan attacks Gosho for 40 hit points! Gosho has 70/100 HP and 0/50 AP
left!
Ivan attacks Gosho for 40 hit points! Gosho has 30/100 HP and 0/50 AP
left!
Gosho attacks Ivan for 40 hit points! Ivan has 100/100 HP and 10/50 AP
left!
Ivan attacks Gosho for 40 hit points! Gosho has 0/100 HP and 0/50 AP
left!
Gosho is dead!
Pesho rests (100 => 100)
Ivan rests (100 => 100)
Ivan attacks Pesho for 40 hit points! Pesho has 100/100 HP and 10/50 AP
left!
Ivan attacks Pesho for 40 hit points! Pesho has 70/100 HP and 0/50 AP
left!
Ivan attacks Pesho for 40 hit points! Pesho has 30/100 HP and 0/50 AP
left!
Ivan attacks Pesho for 40 hit points! Pesho has 0/100 HP and 0/50 AP
left!
Pesho is dead!
Ivan rests (100 => 100)
Ivan rests (100 => 100)
Final stats:
Ivan - HP: 100/100, AP: 10/50, Status: Alive
Gosho - HP: 0/100, AP: 0/50, Status: Dead
Pesho - HP: 0/100, AP: 0/50, Status: Dead
```

Input
JoinParty Java Warrior Gosho

```
JoinParty CSharp Warrior Ivan
Attack Gosho Gosho
PickUpItem Gosho
AddItemToPool InvalidItem
AddItemToPool HealthPotion
UseItem Gosho InvalidItem
UseItem Gosho HealthPotion
PickUpItem InvalidCharacter
Attack Ivan Ivan
Attack Pesho Ivan
Attack Ivan Pesho
Attack A B
Attack Ivan Gosho
Attack Ivan Gosho
Attack Ivan Gosho
Attack Ivan Gosho
EndTurn
EndTurn
```

Output

```
Gosho joined the party!
Ivan joined the party!
Invalid Operation: Cannot attack self!
Invalid Operation: No items left in pool!
Parameter Error: Invalid item "InvalidItem"!
HealthPotion added to pool.
Invalid Operation: Bag is empty!
Invalid Operation: Bag is empty!
Parameter Error: Character InvalidCharacter not found!
Invalid Operation: Cannot attack self!
Parameter Error: Character Pesho not found!
Parameter Error: Character Pesho not found!
Parameter Error: Character A not found!
Ivan attacks Gosho for 40 hit points! Gosho has 100/100 HP and 10/50 AP
left!
Ivan attacks Gosho for 40 hit points! Gosho has 70/100 HP and 0/50 AP
left!
Ivan attacks Gosho for 40 hit points! Gosho has 30/100 HP and 0/50 AP
left!
Ivan attacks Gosho for 40 hit points! Gosho has 0/100 HP and 0/50 AP
left!
Gosho is dead!
Ivan rests (100 => 100)
Ivan rests (100 => 100)
Final stats:
Ivan - HP: 100/100, AP: 50/50, Status: Alive
```

```
Gosho - HP: 0/100, AP: 0/50, Status: Dead
```

Task 4: Bonus (50 Points)

Factories

You know that the keyword new is a bottleneck and we are trying to use it as little as possible. We even try to separate it in classes. These classes are called Factories and the naming convention for them is {TypeOfObject}Factory. You need to have two different factories, one for Characters and one for Items. This is a design pattern and you can read more about it. Factory Pattern. The factories must contain a method (**"CreateCharacter/CreateItem"**), which instantiates objects of that type.

If you try to create a character with an invalid type, throw an **ArgumentException** with a message **"Invalid character type "{type}"!"**.

If you try to create a character with an invalid type, throw an **ArgumentException** with a message **"Invalid item type "{type}"!"**.

Съдържание