**Полета и методи**

**App1**

Тема 3. Полета и свойства упражнения. Зад 3. Списък на служители.

class Employee

**Employee.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App4 |
|  | { |
|  | /// <summary> |
|  | /// Служител |
|  | /// </summary> |
|  | class Employee |
|  | { |
|  | // име |
|  | private string name; |
|  | public string Name |
|  | { |
|  | get { return name; } |
|  | set { name = value; } |
|  | } |
|  | // заплата |
|  | private double salary; |
|  | public double Salary |
|  | { |
|  | get { return salary; } |
|  | set { salary = value; } |
|  | } |
|  | // длъжност |
|  | private string job; |
|  | public string Job |
|  | { |
|  | get { return job; } |
|  | set { job = value; } |
|  | } |
|  | // отдел |
|  | private string department; |
|  | public string Department |
|  | { |
|  | get { return department; } |
|  | set { department = value; } |
|  | } |
|  | // електронна поща |
|  | private string email; |
|  | public string Email |
|  | { |
|  | get { return email; } |
|  | set { email = value; } |
|  | } |
|  | // възраст |
|  | private int age; |
|  | public int Age |
|  | { |
|  | get { return age; } |
|  | set { age = value; } |
|  | } |
|  | } |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App4 |
|  | { |
|  | class Program |
|  | { |
|  | // Главен метод |
|  | static void Main(string[] args) |
|  | { |
|  | // Входни данни |
|  | List<Employee> company = new List<Employee>(); |
|  | int n = int.Parse(Console.ReadLine()); |
|  | while (n > 0) |
|  | { |
|  | var line = Console.ReadLine().Split(); |
|  | switch (line.Count()) |
|  | { |
|  | case 4: |
|  | { |
|  | company.Add(new Employee() |
|  | { |
|  | Name = line[0], |
|  | Salary = double.Parse(line[1]), |
|  | Job = line[2], |
|  | Department = line[3], |
|  | Email = "n/a", |
|  | Age = -1 |
|  | }); |
|  | break; |
|  | } |
|  | case 5: |
|  | { |
|  | int val; |
|  | if (int.TryParse(line[4], out val)) |
|  | company.Add(new Employee() |
|  | { |
|  | Name = line[0], |
|  | Salary = double.Parse(line[1]), |
|  | Job = line[2], |
|  | Department = line[3], |
|  | Email = "n/a", |
|  | Age = val |
|  | }); |
|  | else |
|  | company.Add(new Employee() |
|  | { |
|  | Name = line[0], |
|  | Salary = double.Parse(line[1]), |
|  | Job = line[2], |
|  | Department = line[3], |
|  | Email = line[4], |
|  | Age = -1 |
|  | }); |
|  | break; |
|  | } |
|  | case 6: |
|  | { |
|  | company.Add(new Employee() |
|  | { |
|  | Name = line[0], |
|  | Salary = double.Parse(line[1]), |
|  | Job = line[2], |
|  | Department = line[3], |
|  | Email = line[4], |
|  | Age = int.Parse(line[5]) |
|  | }); |
|  | break; |
|  | } |
|  | } |
|  | n--; |
|  | } |
|  |  |
|  | // Намиране на сума на заплатите по отдели |
|  | var temp = company.GroupBy(a => a.Department).ToDictionary |
|  | ( |
|  | key => key.First().Department, |
|  | value => value.Average(b => b.Salary) |
|  | ).OrderByDescending(c => c.Value); |
|  |  |
|  | // Извеждаме отдела с най-висока средна заплата |
|  | Console.WriteLine("Highest Average Salary: {0}", temp.First().Key); |
|  |  |
|  | // Служителите в този отдел |
|  | var names = company.Where(a => a.Department == temp.First().Key).OrderByDescending(b => b.Salary); |
|  | foreach (var item in names) |
|  | { |
|  | Console.WriteLine("{0} {1} {2} {3} {4} {5}", |
|  | item.Name, item.Salary, item.Job, item.Department, item.Email, item.Age); |
|  | } |
|  |  |
|  | } |
|  | } |
|  | } |

**App2**

Тема 4. Методи упражнения. Зад 2. Най-стария член на фамилията.

class Person

class Family

**Family.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App5 |
|  | { |
|  | /// <summary> |
|  | /// Фамилия |
|  | /// </summary> |
|  | class Family |
|  | { |
|  | // Списък |
|  | private List<Person> members = new List<Person>(); |
|  |  |
|  | // Добавяне |
|  | public void AddMember(Person member) |
|  | { |
|  | members.Add(member); |
|  | } |
|  |  |
|  | // Най-възрастен |
|  | public Person GetOldestMember() |
|  | { |
|  | var list = members.OrderByDescending(a => a.Age).ToList(); |
|  | return list[0]; |
|  | } |
|  | } |
|  | } |

**Person.cs**

|  |
| --- |
| namespace App5 |
|  | { |
|  | /// <summary> |
|  | /// Човек |
|  | /// </summary> |
|  | class Person |
|  | { |
|  | // Име |
|  | private string name; |
|  | public string Name { get { return name; } set { name = value; }} |
|  |  |
|  | // Възраст |
|  | private int age; |
|  | public int Age { get { return age; } set { age = value; } } |
|  | } |
|  |  |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App5 |
|  | { |
|  | class Program |
|  | { |
|  | // Главна функция |
|  | static void Main(string[] args) |
|  | { |
|  | Family family = new Family(); |
|  | // Входни данни |
|  | 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--; |
|  | } |
|  | // Намиране на най-възрастния |
|  | var oldest = family.GetOldestMember(); |
|  | Console.WriteLine("Oldest: {0} {1}", oldest.Name, oldest.Age); |
|  | } |
|  | } |
|  | } |

**App3**

Тема 4. Методи. Зад 3. Статистическо проучване

class Person

class Family

**Family.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App3 |
|  | { |
|  | class Family |
|  | { |
|  | private List<Person> people = new List<Person>(); |
|  | public void AddMember(Person member) |
|  | { |
|  | people.Add(member); |
|  | } |
|  | public void Over30() |
|  | { |
|  | List<Person> over30 = new List<Person>(); |
|  | for (int i = 0; i < people.Count; i++) |
|  | { |
|  | if (people[i].Age > 30) |
|  | over30.Add(people[i]); |
|  | } |
|  | over30 = over30.OrderBy(x => x.Name).ToList(); |
|  | foreach (var person in over30) |
|  | { |
|  | Console.WriteLine($"{person.Name} - {person.Age}"); |
|  | } |
|  | } |
|  | } |
|  | } |

**Person.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App3 |
|  | { |
|  | class Person |
|  | { |
|  | private string name; |
|  | private int age; |
|  | public string Name |
|  | { |
|  | get { return name; } |
|  | set { name = value; } |
|  | } |
|  | public int Age |
|  | { |
|  | get { return age; } |
|  | set { age = value; } |
|  | } |
|  | } |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App3 |
|  | { |
|  | class Program |
|  | { |
|  | // Главен метод |
|  | static void Main(string[] args) |
|  | { |
|  | int n = int.Parse(Console.ReadLine()); |
|  | Family semeistvo = new Family(); |
|  | for (int i = 0; i < n; i++) |
|  | { |
|  | string[] input = Console.ReadLine().Split(' ').ToArray(); |
|  | semeistvo.AddMember |
|  | ( |
|  | new Person { Name = input[0], Age = int.Parse(input[1]) } |
|  | ); |
|  | } |
|  | semeistvo.Over30(); |
|  | } |
|  | } |
|  | } |

**App4**

Тема 4. Методи. Зад 4. Разликата в дни между две дати

class DateModifier

**DateModifier.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  | using System.Globalization; |
|  |  |
|  | namespace App4 |
|  | { |
|  | class DateModifier |
|  | { |
|  | private DateTime first, second; |
|  | public DateTime First |
|  | { |
|  | get { return first; } |
|  | set { first = value; } |
|  | } |
|  | public DateTime Second |
|  | { |
|  | get { return second; } |
|  | set { second = value; } |
|  | } |
|  | public void InIt(string d1, string d2) |
|  | { |
|  | this.First = DateTime.ParseExact(d1, "yyyy MM dd", CultureInfo.InvariantCulture); |
|  | this.Second = DateTime.ParseExact(d2, "yyyy MM dd", CultureInfo.InvariantCulture); |
|  | } |
|  | public int Difference() |
|  | { |
|  | int days = Math.Abs(this.First.Subtract(this.Second).Days); |
|  | return days; |
|  | } |
|  | }  } |
| **Program.cs** |  |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App4 |
|  | { |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  | string first = Console.ReadLine(); |
|  | string second = Console.ReadLine(); |
|  | DateModifier date = new DateModifier(); |
|  | date.InIt(first, second); |
|  | Console.WriteLine(date.Difference()); |
|  | } |
|  | } |
|  | } |

**App5**

Тема 5. Конструктори. Зад 1. Дефиниране на класа Човек + Зад 2. Конструктори за класа Човек

class Person

class BankAccount

**BankAccount.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App5 |
|  | { |
|  | class BankAccount |
|  | { |
|  | // Иденфикатор |
|  | private int id; |
|  | public int ID |
|  | { |
|  | get { return id; } |
|  | set { id = value; } |
|  | } |
|  |  |
|  | // Баланс |
|  | private double balance; |
|  | public double Balance |
|  | { |
|  | get { return balance; } |
|  | set { balance = value; } |
|  | } |
|  | } |
|  |  |
|  | } |

**Person.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App5 |
|  | { |
|  | class Person |
|  | { |
|  | private string name; |
|  | public string Name |
|  | { |
|  | set { name = value; } |
|  | get { return name; } |
|  | } |
|  | private int age; |
|  | public int Age |
|  | { |
|  | set { age = value; } |
|  | get { return age; } |
|  | } |
|  | private List<BankAccount> accounts; |
|  |  |
|  | public Person() |
|  | { |
|  | this.name = "No name"; |
|  | this.age = 1; |
|  | } |
|  | public Person(int age) |
|  | { |
|  | this.name = "No name"; |
|  | this.age = age; |
|  | } |
|  | public Person(string name, int age) |
|  | { |
|  | this.name = name; |
|  | this.age = age; |
|  | this.accounts = 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 accounts.Sum(x => x.Balance); |
|  | } |
|  | } |
|  |  |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App5 |
|  | { |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  | string name = Console.ReadLine(); |
|  | int age = int.Parse(Console.ReadLine()); |
|  |  |
|  | Person First = new Person(); |
|  | Person Second = new Person(age); |
|  | Person Third = new Person(name, age); |
|  |  |
|  | Console.WriteLine("{0} {1}", First.Name, First.Age); |
|  | Console.WriteLine("{0} {1}", Second.Name, Second.Age); |
|  | Console.WriteLine("{0} {1}", Third.Name, Third.Age); |
|  |  |
|  | } |
|  | } |
|  | } |

**App6**

Тема 5. Конструктори. Зад 3. Сурови данни

class Car

class CarEngine

class Cargo

class Tire

**Car.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Text; |
|  |  |
|  |  |
|  | class Car |
|  | { |
|  | public string Model { get; set; } |
|  |  |
|  | public CarEngine Engine { get; set; } |
|  |  |
|  | public Cargo Cargo { get; set; } |
|  |  |
|  | public Tire Tire1 { get; set; } |
|  |  |
|  | public Tire Tire2 { get; set; } |
|  |  |
|  | public Tire Tire3 { get; set; } |
|  |  |
|  | public Tire Tire4 { get; set; } |
|  |  |
|  | public Car(string model, int engineSpeed, int enginePower, int cargoWeight, string cargoType, |
|  | double tire1Pressure, int tire1Age, double tire2Pressure, int tire2Age, |
|  | double tire3Pressure, int tire3Age, double tire4Pressure, int tire4Age) |
|  | { |
|  | this.Model = model; |
|  | this.Engine = new CarEngine(engineSpeed, enginePower); |
|  | this.Cargo = new Cargo(cargoWeight, cargoType); |
|  | this.Tire1 = new Tire(tire1Pressure, tire1Age); |
|  | this.Tire2 = new Tire(tire2Pressure, tire2Age); |
|  | this.Tire3 = new Tire(tire3Pressure, tire3Age); |
|  | this.Tire4 = new Tire(tire4Pressure, tire4Age); |
|  | } |
|  |  |
|  | } |

**CarEngine.cs**

|  |
| --- |
| class CarEngine |
|  | { |
|  | public int Speed { get; set; } |
|  |  |
|  | public int Power { get; set; } |
|  |  |
|  | public CarEngine(int speed, int power) |
|  | { |
|  | this.Speed = speed; |
|  | this.Power = power; |
|  | } |
|  | } |

**Cargo.cs**

|  |
| --- |
| class Cargo |
|  | { |
|  | public int Weight { get; set; } |
|  |  |
|  | public string Type { get; set; } |
|  |  |
|  | public Cargo(int weight, string type) |
|  | { |
|  | this.Weight = weight; |
|  | this.Type = type; |
|  | } |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  |  |
|  | namespace OOPBasics\_Burgasko |
|  | { |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  | var n = int.Parse(Console.ReadLine()); |
|  | var cars = new List<Car>(); |
|  |  |
|  | for (int i = 0; i < n; i++) |
|  | { |
|  | var carInfo = Console.ReadLine().Split(' '); |
|  | var carModel = carInfo[0]; |
|  | var carEngineSpeed = int.Parse(carInfo[1]); |
|  | var carEnginePower = int.Parse(carInfo[2]); |
|  | var carCargoWeight = int.Parse(carInfo[3]); |
|  | var carCargoType = carInfo[4]; |
|  | var carTire1Pressure = double.Parse(carInfo[5]); |
|  | var carTire1Age = int.Parse(carInfo[6]); |
|  | var carTire2Pressure = double.Parse(carInfo[7]); |
|  | var carTire2Age = int.Parse(carInfo[8]); |
|  | var carTire3Pressure = double.Parse(carInfo[9]); |
|  | var carTire3Age = int.Parse(carInfo[10]); |
|  | var carTire4Pressure = double.Parse(carInfo[11]); |
|  | var carTire4Age = int.Parse(carInfo[12]); |
|  |  |
|  | cars.Add(new Car(carModel, carEngineSpeed, carEnginePower, carCargoWeight, carCargoType, |
|  | carTire1Pressure, carTire1Age, carTire2Pressure, carTire2Age, |
|  | carTire3Pressure, carTire3Age, carTire4Pressure, carTire4Age)); |
|  | } |
|  |  |
|  | var outputRequirement = Console.ReadLine(); |
|  |  |
|  | IEnumerable<Car> result; |
|  |  |
|  | if (outputRequirement == "fragile") |
|  | { |
|  | result = cars.Where(c => c.Cargo.Type == outputRequirement) |
|  | .Where(c => c.Tire1.Pressure < 1 || c.Tire2.Pressure < 1 || |
|  | c.Tire3.Pressure < 1 || c.Tire4.Pressure < 1); |
|  | } |
|  | else |
|  | { |
|  | result = cars.Where(c => c.Cargo.Type == outputRequirement) |
|  | .Where(c => c.Engine.Power > 250); |
|  | } |
|  |  |
|  | Console.WriteLine(String.Join(Environment.NewLine, result.Select(c => c.Model))); |
|  | } |
|  | } |
|  | } |

**Tire.cs**

|  |
| --- |
| class Tire |
|  | { |
|  | public double Pressure { get; set; } |
|  |  |
|  | public int Age { get; set; } |
|  |  |
|  | public Tire(double pressure, int age) |
|  | { |
|  | this.Pressure = pressure; |
|  | this.Age = age; |
|  | } |
|  | } |

**App7**

Тема 6. Допълнителна подготовка. Зад 1. Пътувания с коли.

class Car

**Car.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App7 |
|  | { |
|  | class Car |
|  | { |
|  | // Name |
|  | private string name; |
|  | public string Name |
|  | { |
|  | get { return name; } |
|  | set { name = value; } |
|  | } |
|  |  |
|  | // Fuel |
|  | private float fuel; |
|  | public float Fuel |
|  | { |
|  | get { return fuel; } |
|  | set { fuel = value; } |
|  | } |
|  |  |
|  | // perkm |
|  | private float perkm; |
|  | public float Perkm |
|  | { |
|  | get { return perkm; } |
|  | set { perkm = value; } |
|  | } |
|  |  |
|  | // The Magic |
|  | 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); |
|  | } |
|  |  |
|  | } |
|  | } |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App7 |
|  | { |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  | // Car |
|  | 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]) |
|  | } |
|  | ); |
|  | } |
|  |  |
|  | // Commands |
|  | var dik = new Dictionary<string, int>(); |
|  | var line = ""; |
|  | do |
|  | { |
|  | line = Console.ReadLine(); |
|  | if (line != "End") |
|  | { |
|  | var split = line.Split(); |
|  | if (dik.ContainsKey(split[1])) |
|  | { |
|  | var km = dik[split[1]] + int.Parse(split[2]); |
|  | dik[split[1]] = km; |
|  |  |
|  | } |
|  | else dik.Add(split[1], int.Parse(split[2])); |
|  | } |
|  | } |
|  | while (line != "End"); |
|  |  |
|  | // Process |
|  | foreach (var item in dik) |
|  | { |
|  | var currentCar = cars.Where(a => a.Name == item.Key).First(); |
|  | currentCar.TheMagic(item.Value); |
|  | } |
|  | } |
|  | } |
|  | } |

**App8**

Тема 6. Допълнителна подготовка. Зад 2. Застъпване на правоъгълници.

class Rectangle

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  |  |
|  | namespace Rectangle |
|  | { |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  | List<Rectangle> info = new List<Rectangle>(); |
|  |  |
|  | // Input |
|  | 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]) |
|  | } |
|  | ); |
|  | } |
|  |  |
|  | // Output |
|  | 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"); |
|  | } |
|  | } |
|  | } |

**Rectangle.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  |  |
|  | namespace Rectangle |
|  | { |
|  | class Rectangle |
|  | { |
|  | // id |
|  | private string id; |
|  | public string ID |
|  | { |
|  | get { return id; } |
|  | set { id = value; } |
|  | } |
|  |  |
|  | // width |
|  | private int width; |
|  | public int Width |
|  | { |
|  | get { return width; } |
|  | set { width = value; } |
|  | } |
|  |  |
|  | // height |
|  | private int heigh; |
|  | public int Heigh |
|  | { |
|  | get { return heigh; } |
|  | set { heigh = value; } |
|  | } |
|  | // horizontally |
|  | private int horizontally; |
|  | public int Horizontally |
|  | { |
|  | get { return horizontally; } |
|  | set { horizontally = value; } |
|  | } |
|  |  |
|  | // vertically |
|  | private int vertically; |
|  | public int Vertically |
|  | { |
|  | get { return vertically; } |
|  | set { vertically = value; } |
|  | } |
|  | } |
|  | } |

**App9**

Тема 6. Допълнителна подготовка. Зад 3. Продавач на коли.

class Car

class Engine

**App10**

Тема 6. Допълнителна подготовка. Зад 4. Треньор на покемони.

class Trainer

class Pokemon

**App11**

Тема 6. Допълнителна подготовка. Зад 5. Google.

class Google

|  |
| --- |
| **Pokemon.cs**  using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App11 |
|  | { |
|  | 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}"; |
|  | } |
|  | } |
|  | } |

**Car.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App11 |
|  | { |
|  | 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"; |
|  | } |
|  | } |
|  | } |

**Child.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App11 |
|  | { |
|  | 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}"; |
|  | } |
|  | } |
|  | } |

**Company.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App11 |
|  | { |
|  | 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"; |
|  | } |
|  | } |
|  | } |

**Parent.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App11 |
|  | { |
|  | 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}"; |
|  | } |
|  | } |
|  | } |

**Person.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App11 |
|  | { |
|  | 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; } |
|  | } |
|  | } |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace App11 |
|  | { |
|  | 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); |
|  | } |
|  | } |
|  | } |

**App12**

Тема 6. Допълнителна подготовка. Зад 6. Родословно дърво.

class FamilyTree

**FamilyInfo.cs**

|  |
| --- |
| using System.Collections.Generic; |
|  |  |
|  | public 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>(); |
|  | } |
|  | } |

**FamilyTree.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Globalization; |
|  | using System.Linq; |
|  |  |
|  | public 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; |
|  | } |
|  | } |

**Person.cs**

|  |
| --- |
| using System; |
|  |  |
|  | public 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")}"; |
|  | } |
|  | } |

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Globalization; |
|  | using System.Linq; |
|  |  |
|  | 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)); |
|  | } |
|  | } |