Lab: Encapsulation

Problems for the "C# OOP" course @ SoftUni".

You can check your solutions here: https://judge.softuni.org/Contests/1497/Encapsulation-Lab

1. Sort People by Name and Age

NOTE: You need a public **StartUp** class with the namespace **PersonsInfo**.

Create a class **Person**, which should have **public** properties with **private** setters for:

- **FirstName: string**
- **LastName: string**
- **Age: int**
- **ToString(): string - override**

You should be able to use the class like this:

StartUp.cs

```
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()));
}
```

Examples

Input	Output
5 Brandi Anderson 65 Andrew Williams 57 Newton Holland 27 Andrew Clark 44 Brandi Scott 35	Andrew Clark is 44 years old. Andrew Williams is 57 years old. Brandi Scott is 35 years old. Brandi Anderson is 65 years old. Newton Holland is 27 years old.

Solution

Create a **new class** and ensure **proper naming**. Define the **public** properties:

Person.cs

```
public class Person
{
    public string FirstName { get; private set; }

    public string LastName { get; private set; }

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

Create a constructor for **Person**, which takes 3 parameters **firstName**, **lastName**, **age**:

Person.cs

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

Override **ToString()** method:

Person.cs

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

2. Salary Increase

NOTE: You need a public **StartUp** class with the namespace **PersonsInfo**. **Refactor the project from the last task.**

Create objects of the class **Person**. Read their **name**, **age**, and **salary** from the console. Read the percentage of the bonus to every **Person's salary**. People younger than **30 get half the increase**. Expand **Person** from the previous task.

New **properties** and **methods:**

- **Salary: decimal**
- **IncreaseSalary(decimal percentage)**

You should be able to use the class like this:

StartUp.cs

```
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]),
                            decimal.Parse(cmdArgs[3]));

    persons.Add(person);
}
var parcentage = decimal.Parse(Console.ReadLine());
persons.ForEach(p => p.IncreaseSalary(parcentage));
persons.ForEach(p => Console.WriteLine(p.ToString()));
```

Examples

Input	Output
5 Andrew Williams 65 2200 Newton Holland 57 3333 Rachelle Nelson 27 600 Brandi Scott 44 666.66 George Miller 35 559.4 20	Andrew Williams receives 2640.00 leva. Newton Holland receives 3999.60 leva. Rachelle Nelson receives 660.00 leva. Brandi Scott receives 799.99 leva. George Miller receives 671.28 leva.

Solution

Add a new **public** property for **salary** and **refactor the constructor**. Add a new **method**, which will **update salary** with a bonus:

Person.cs

```
public void IncreaseSalary(decimal percentage)
{
    if(this.Age > 30)
    {
        this.Salary += this.Salary * percentage / 100;
    }
    else
    {
        this.Salary += this.Salary * percentage / 200;
    }
}
```

Refactor the **ToString()** method for this task.

3. Validation of Data

NOTE: You need a public **StartUp** class with the namespace **PersonsInfo**. The skeleton from the previous task could be used.

Expand **Person** with proper **validation** for every **field**:

- **Name** must be at **least 3 symbols**
- **Age** must **not** be **zero or negative**
- **Salary can't** be **less than 460 (decimal)**

If some of the properties are **NOT valid** throw **ArgumentExeption** with the following messages:

- **"Age cannot be zero or a negative integer!"**
- **"First name cannot contain fewer than 3 symbols!"**
- **"Last name cannot contain fewer than 3 symbols!"**
- **"Salary cannot be less than 650 leva!"**

Examples

Input	Output
5 Andrew Williams -6 2200 B Gomez 57 3333 Carolina Richards 27 670 Gilbert H 44 666.66 Joshua Anderson 35 300 20	Age cannot be zero or a negative integer! First name cannot contain fewer than 3 symbols! Last name cannot contain fewer than 3 symbols! Salary cannot be less than 650 leva! Carolina Richards receives 737.00 leva.

Solution

Add validation to all of the setters in **Person**. Validation may look like this or something similar:

Person.cs

```
public decimal Salary
{
    get
    {
        return this.salary;
    }
    private set
    {
        if (value < 650)
        {
            throw new ArgumentException("Salary cannot be less than 650 leva!");
        }
    }
}
```

4. First and Reserve Team

NOTE: You need a public **StartUp** class with the namespace **PersonsInfo**. The skeleton from the previous task could be used.

Create a **Team** class. Add to this team all of the people you have received. Those who are **younger than 40** go to the **first team**, **others** go to the **reserve team**. At the end print the **sizes of the first** and the **reserved team**.

The class should have **private fields** for:

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

The class should have **constructors**:

- **`Team(string name)`**

The class should also have **public properties** for:

- **`FirstTeam: List<Person> (read-only!)`**
- **`ReserveTeam: List<Person> (read-only!)`**

And a **method** for **adding players**:

- **`AddPlayer(Person person): void`**

You should be able to use the class like this:

StartUp.cs

```
Team team = new Team("SoftUni");

foreach (Person person in persons)
{
    team.AddPlayer(person);
}
```

You should **NOT** be able to use the class like this:

StartUp.cs

```
Team team = new Team("SoftUni");

foreach (Person person in persons)
{
    if(person.Age < 40)
    {
        team.FirstTeam.Add(person);
    }
    else
    {
        team.ReserveTeam(person);
    }
}
```

Examples

Input	Output
5 Andrew Williams 20 2200 Newton Holland 57 3333 Rachelle Nelson 27 600 Grigor Dimitrov 25 666.66 Brandi Scott 35 555	First team has 4 players. Reserve team has 1 players.

Solution

Add a new class **Team**. Its fields and **constructor** should look like

Team.cs

```
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>();
}
```

Properties for **FirstTeam** and **ReserveTeam** have only **getters**:

Team.cs

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

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

There will be only **one method**, which **adds players** to teams:

Team.cs

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