# Lab: Encapsulation

## Sort Persons by Name and Age

Create a class **Person**, which should have **private** fields for:

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

You should be able to use the class like this:

|  |
| --- |
| Program.cs |
| public static void Main()  {  var lines = 5;  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** |
| Anna Persson 34  Magnus Petterson 65  Johanna Eriksson 35  Peter Forsberg 27  Erika Samuelsson 35 | Anna Persson is 34 years old.  Magnus Petterson is 65 years old.  Johanna Eriksson is 35 years old.  Peter Forsberg is 27 years old.  Erika Samuelsson is 35 years old. |

### Solution

Create a **new class** and ensure **proper naming**. Define the **private** fields:



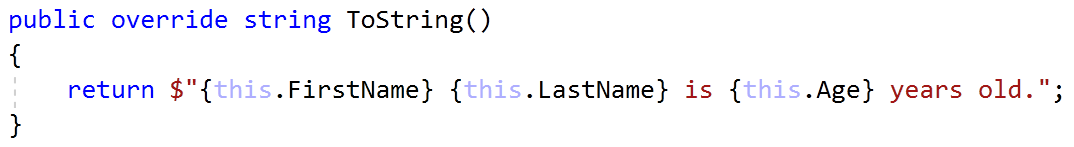
Create a constructor for Person, which takes 3 parameters firstName, lastName, age:



Create properties for these fields, which are as strictly as possible:



Override **ToString()** method:



## Salary Increase

**Refactor project from last task.**

Read person with their names, age and salary. Read percent bonus to every person salary. People younger than 30 **get half the increase**. Expand **Person** from the previous task.

New **fields** and **methods:**

* **salary**: **decimal**
* **IncreaseSalary**(**decimal** **percentage**)

You should be able to use the class like this:

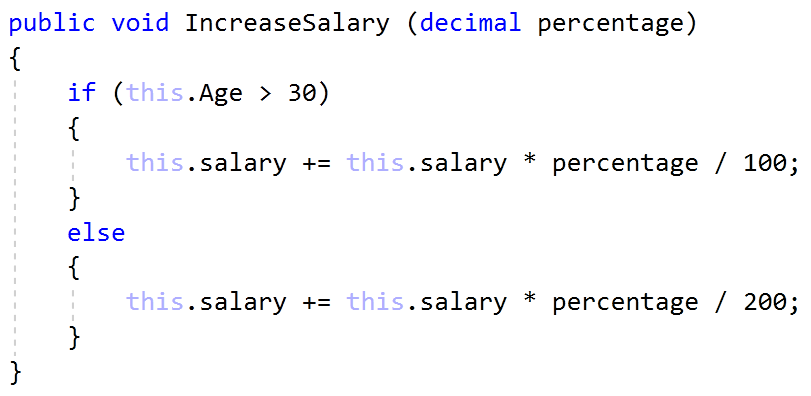
|  |
| --- |
| program.cs |
| var lines = int.Parse(Console.ReadLine());  var persons = new List<Person>();  for (int i = 0; i < lines; i++)  {  var cmdArgs = Console.ReadLine().Split();  var person = new Person(cmdArgs[0],  cmdArgs[1],  int.Parse(cmdArgs[2]),  decimal.Parse(cmdArgs[3]));  persons.Add(person);  }  var bonus = decimal.Parse(Console.ReadLine());  persons.ForEach(p => p.IncreaseSalary(bonus));  persons.ForEach(p => Console.WriteLine(p.ToString())); |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Ida Svensson 65 2200  Berit Dahl 57 3333  Bert Lewinsson 27 600  Anna Hamren 44 666.66  Jacob Andersson 35 559.4 | Ida Svensson receives 2640.00 dollars.  Berit Dahl receives 3999.60 dollars.  Bert Lewinsson receives 660.00 dollars.  Anna Hamren receives 799.99 dollars.  Jacob Andersson receives 671.28 dollars. |

### Solution

Add new **private** field for **salary** and **refactor constructor**. Add new **method**, which will **update** salary with bonus



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

## Validation of Data

Expand Person with proper validation for every field:

* **Names must be at least 3 symbols**
* **Age must not be zero or negative**
* **Salary can't be less than 460.0**

Print proper messages to the user:

* **“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 460 dollar!”**

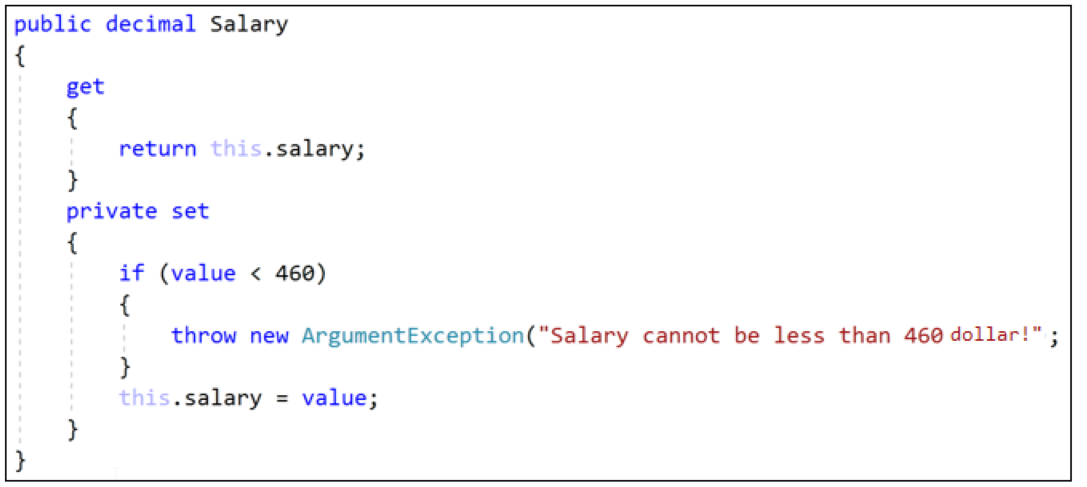
Use ArgumentExeption with messages from example.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Anna Ivanov -6 2200  B Dahl 57 3333  Hanna Ma 27 600  Asen H 44 666  Mikael Johansson 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 460 dollars!  Boris Magnusson gets 660.00 dollars. |

### Solution

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



## First and Reserve Team

Create a Team class. Add to this team all people you read. All people younger than 40 go on the first team, others go on the reverse team. At the end print the first and reserve team sizes.

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

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

The class should have **constructors**:

* Team(string name)

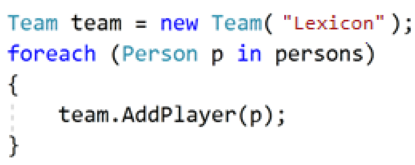
The class should also have **public properties** for:

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

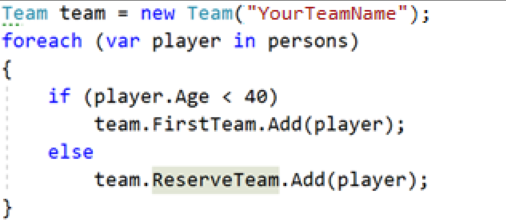
And a method for adding players:

* AddPlayer(Person person): void

You should be able to use the class like this:



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

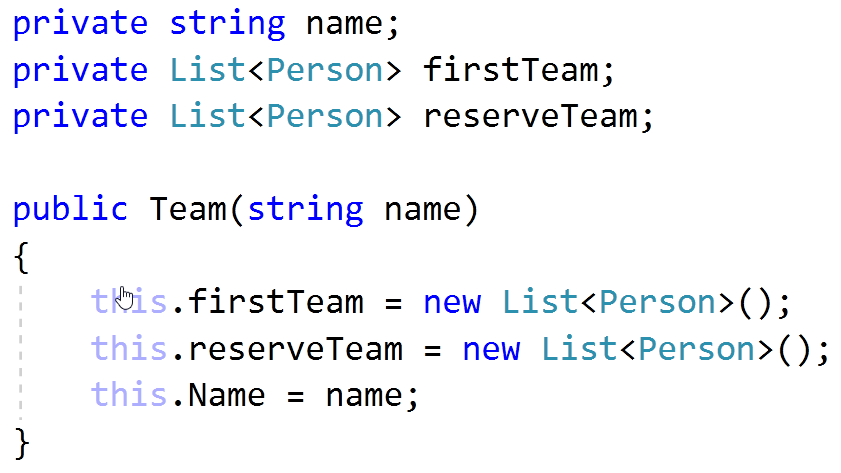
 

### Examples

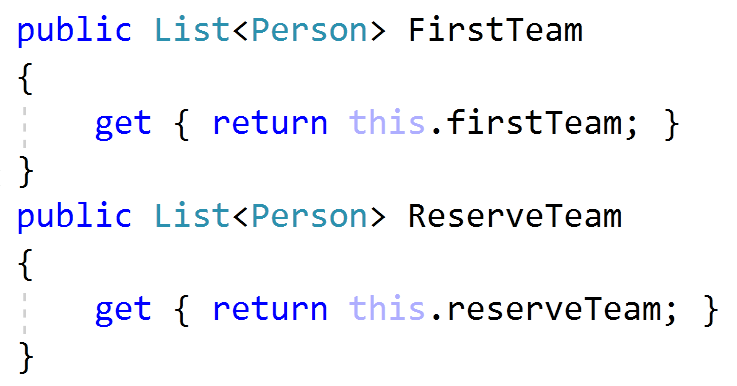
|  |  |
| --- | --- |
| **Input** | **Output** |
| Johanna Eriksson 20 2200  Bert 57 3333  Hanna Magnusson 27 600  Gregor Jackson 25 666.66  Boiko Angelov 35 555 | First team has 4 players.  Reserve team has 1 players. |

### Solution

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



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



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

