1. C#: Team Interface

1. C#: Team Interface

Implement inheritance as described below.

Create a class Team that has the following:

1. A member variable *teamName* [string]
2. A member variable *noOfPlayers* [integer]
3. A constructor function:
   1. It takes 2 parameters and assigns them to *teamName* and *noOfPlayers* respectively.
4. A member function *AddPlayer(count)*:
   1. It takes an integer *count* as a parameter and increases *noOfPlayers* by *count*.
5. A member function *RemovePlayer(count)*:
   1. It takes an integer *count* as a parameter and tries to decrease *noOfPlayers* by *count*.
   2. If decreasing makes *noOfPlayers* negative, then this function simply returns false.
   3. Else, decrease *noOfPlayers* by *count* and return true.

Create a class Subteam that inherits from the above class Team. It has the following:

1. A constructor function:
   1. It takes 2 parameters, *teamName* and *noOfPlayers*, and calls the base class constructor with these parameters.
2. A member function *ChangeTeamName(name)*:
   1. It takes a string *name* as a parameter and changes *teamName* to *name*.

Note: Declare all the members as public so that they are accessible by the stubbed code.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the function calls. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the stubbed code.

be printed to the standard output by the stubbed code.

Language: C# Environment Autocomplete Ready

```
using System;…
    public class Team {
        public string teamName;
        public int noOfPlayers;

        public Team(string teamName, int noOfPlayers){
            this.teamName = teamName;
            this.noOfPlayers = noOfPlayers;
        }
        public void AddPlayer(int count){
            noOfPlayers += count;
        }
        public bool RemovePlayer(int count){
            if(noOfPlayers < count){
                return false;
            }
            noOfPlayers -= count;
            return true;
        }
    }

    public class Subteam: Team{
    public Subteam(string teamName, int noOfPlayers): base(teamName, noOfPlayers){

    }

    public void ChangeTeamName(string name){
        teamName = name;

    }
}


    class Solution {…
```

Line: 7 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

▼ Input Format For Custom Testing

The first line contains a string, *teamName*, and integer, *noOfPlayers*, denoting the team name and the initial number of players in the team, respectively.
The second line contains an integer, *count*, which is the parameter for the *AddPlayer* function.
The third line contains an integer, *count*, which is the parameter for the *RemovePlayer* function.
The fourth line contains a string, *name*, which is the parameter for *ChangeTeamName* function.

▼ Sample Case 0

Sample Input For Custom Testing

```
OldTeam 2
3
4
NewTeam
```

Sample Output

```
Team OldTeam created
Current number of players in team OldTeam is 2
New number of players in team OldTeam is 5
Current number of players in team OldTeam is 5
New number of players in team OldTeam is 1
Team name of team OldTeam changed to NewTeam
```

Explanation

First, a team is created with *teamName* as 'OldTeam' and *noOfPlayers* as 2. Then, the *AddPlayer* function is called with parameter 3, so the new *noOfPlayers* becomes 5. Then, the *RemovePlayer* function is called with parameter 4, so the new *noOfPlayers* becomes 1. Finally, the *ChangeTeamName* function is called with parameter 'NewTeam', which changes *teamName* to 'NewTeam'.

▼ Sample Case 1

Sample Input For Custom Testing

```
Champions 2
1
2
IPL
```

Sample Output

Language: C# Environment Autocomplete Ready

```
        return true;
        }
    }

    public class Subteam: Team{
    public Subteam(string teamName, int noOfPlayers): base(teamName, noOfPlayers){

    }

    public void ChangeTeamName(string name){
        teamName = name;

    }
}
```

Line: 7 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Input (stdin) Run as Custom Input Download

```
1 OldTeam 2
2 3
3 4
4 NewTeam
```

Your Output (stdout)

```
1 Team OldTeam created
2 Current number of players in team OldTeam is 2
3 New number of players in team OldTeam is 5
4 Current number of players in team OldTeam is 5
5 New number of players in team OldTeam is 1
6 Team name of team OldTeam changed to NewTeam
```

```
3
4
NewTeam
```

Sample Output

```
Team OldTeam created
Current number of players in team OldTeam is 2
New number of players in team OldTeam is 5
Current number of players in team OldTeam is 5
New number of players in team OldTeam is 1
Team name of team OldTeam changed to NewTeam
```

Explanation

First, a team is created with *teamName* as 'OldTeam' and *noOfPlayers* as 2. Then, the *AddPlayer* function is called with parameter 3, so the new *noOfPlayers* becomes 5. Then, the *RemovePlayer* function is called with parameter 4, so the new *noOfPlayers* becomes 1. Finally, the *ChangeTeamName* function is called with parameter 'NewTeam', which changes *teamName* to 'NewTeam'.

▼ Sample Case 1

Sample Input For Custom Testing

```
Champions 2
1
2
IPL
```

Sample Output

```
Team Champions created
Current number of players in team Champions is 2
New number of players in team Champions is 3
Current number of players in team Champions is 3
New number of players in team Champions is 1
Team name of team Champions changed to IPL
```

Explanation

First, a team is created with *teamName* as 'Champions' and *noOfPlayers* as 2. Then, the *AddPlayer* function is called with parameter 1, so the new *noOfPlayers* becomes 3. Then, the *RemovePlayer* function is called with parameter 2, so the new *noOfPlayers* becomes 1. Finally, the *ChangeTeamName* function is called with parameter 'IPL', which changes *teamName* to 'IPL'.

Language: C# | Environment | Autocomplete Loading...

```
            return true;
        }
    }

    public class Subteam: Team{
    public Subteam(string teamName, int noOfPlayers): base(teamName, noOfPlayers){

    }

    public void ChangeTeamName(string name){
        teamName = name;

    }
}
```

Line: 7 Col:

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

```
Champions 2
1
2
IPL
```

Your Output (stdout)

```
Team Champions created
Current number of players in team Champions is 2
New number of players in team Champions is 3
Current number of players in team Champions is 3
New number of players in team Champions is 1
Team name of team Champions changed to IPL
```

1. C#: Team Interface

Implement inheritance as described below.

Create a class Team that has the following:

1. A member variable *teamName* [string]
2. A member variable *noOfPlayers* [integer]
3. A constructor function:
   1. It takes 2 parameters and assigns them to *teamName* and *noOfPlayers* respectively.
4. A member function *AddPlayer(count)*:
   1. It takes an integer *count* as a parameter and increases *noOfPlayers* by *count*.
5. A member function *RemovePlayer(count)*:
   1. It takes an integer *count* as a parameter and tries to decrease *noOfPlayers* by *count*.
   2. If decreasing makes *noOfPlayers* negative, then this function simply returns false.
   3. Else, decrease *noOfPlayers* by *count* and return true.

Create a class Subteam that inherits from the above class Team. It has the following:

1. A constructor function:
   1. It takes 2 parameters, *teamName* and *noOfPlayers*, and calls the base class constructor with these parameters.
2. A member function *ChangeTeamName(name)*:
   1. It takes a string *name* as a parameter and changes *teamName* to *name*.

Note: Declare all the members as public so that they are accessible by the stubbed code.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the function calls. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the stubbed code.

Language: C# | Environment | Autocomplete Ready

```
using System;…
    public class Team {
        public string teamName;
        public int noOfPlayers;

        public Team(string teamName, int noOfPlayers){
            this.teamName = teamName;
            this.noOfPlayers = noOfPlayers;
        }
        public void AddPlayer(int count){
            noOfPlayers += count;
        }
        public bool RemovePlayer(int count){
            if(noOfPlayers < count){
                return false;
            }
            noOfPlayers -= count;
            return true;
        }
    }

    public class Subteam: Team{
    public Subteam(string teamName, int noOfPlayers): base(teamName, noOfPlayers){

    }

    public void ChangeTeamName(string name){
        teamName = name;

    }
}


    class Solution {…
```

Line: 7 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

2. C#: Employees Management

2. C#: Employees Management

Given a list of data, implement the following 3 methods using LINQ in a class that allows for managing employees:

1. *AverageAgeForEachCompany* - calculates the average age of employees for each unique company and returns the results as a **Dictionary<string, int>** sorted by key, where Key[string] is the unique company name, and Value[int] is the average age of employees in this company. The average age is rounded to the nearest whole number.
2. *CountOfEmployeesForEachCompany* - calculates the count of employees for each unique company and returns the results as a **Dictionary<string, int>** sorted by key, where Key[string] is the unique company name, and Value[int] is the count of employees in this company.
3. *OldestAgeForEachCompany* - finds the oldest aged employee for each unique company and returns the results as a **Dictionary<string, Employee>** sorted by key, where Key[string] is the unique company name, and Value[Employee] is the oldest employee in this company.

Here is the description of the Employee class:

- *FirstName* [string] - the first name of the employee.
- *LastName*[string] - the last name of the employee.
- *Company* [string] - the company of the employee.
- *Age* [int] - the age of the employee.

Your implementation of the class will be tested by a stubbed code on several input files. The input file contains parameters for the function calls (i.e., the employee data). The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the stubbed code.

▸ **Input Format For Custom Testing**

▸ **Sample Case 0**

▾ **Input Format For Custom Testing**

The first line contains an integer, *n*, denoting the number of employees on which operations have to be performed. Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains space-separated strings, such that the first of them is the first name of the employee, the second is the last name of the employee, the third is the company of the employee, and the fourth is the age of the employee, respectively.

▾ **Sample Case 0**

Sample Input For Custom Testing

```
12
Ainslee Ginsie Galaxy 28
Libbey Apdell Starbucks 44
Illa Stebbings Berkshire 49
Laina Sycamore Berkshire 20
Abbe Parnell Amazon 20
Ludovika Reveley Berkshire 30
Rene Antos Galaxy 44
Vinson Beckenham Berkshire 45
Reed Lynock Amazon 41
Wyndham Bamfield Berkshire 34
Loraine Sappson Amazon 49
Abbe Antonutti Starbucks 47
```

Sample Output

```
The average age for company Amazon is 37
The average age for company Berkshire is 36
The average age for company Galaxy is 36
The average age for company Starbucks is 46
The count of employees for company Amazon is 3
The count of employees for company Berkshire is
5
The count of employees for company Galaxy is 2
The count of employees for company Starbucks is
2
The oldest employee of company Amazon is
Loraine Sappson having age 49
The oldest employee of company Berkshire is
Illa Stebbings having age 49
The oldest employee of company Galaxy is Rene
Antos having age 44
The oldest employee of company Starbucks is
Abbe Antonutti having age 47
```

Language: C# | Environment | Autocomplete Ready

```
using System;
public class Solution
{
    public static Dictionary<string, int> AverageAgeForEachCompany(List<Employee> employees)
    {
        var result = new Dictionary<string, int>();
        foreach( var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
        {
            result.Add(company, (int)Math.Round(employees.Where(x => x.Company == company).Average(y => y.Age), 0)
);
        }
        return result;

    }

    public static Dictionary<string, int> CountOfEmployeesForEachCompany(List<Employee> employees)
    {
        var result = new Dictionary<string, int>();
        foreach(var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
        {
            result.Add(company, (int)employees.Where(x => x.Company == company).Count());
        }
        return result;
    }

    public static Dictionary<string, Employee> OldestAgeForEachCompany(List<Employee> employees)
    {
        var result = new Dictionary<string, Employee>();
        foreach(var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
        {
            result.Add(company, employees.Where(x => x.Company == company).OrderByDescending(y => y.Age).First());
        }
        return result;
    }
    public static void Main()
```

Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Language: C# | Environment | Autocomplete Ready

```
            result.Add(company, (int)employees.Where(x => x.Company == company).Count());
        }
        return result;
    }

    public static Dictionary<string, Employee> OldestAgeForEachCompany(List<Employee> employees)
    {
        var result = new Dictionary<string, Employee>();
        foreach(var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
        {
            result.Add(company, employees.Where(x => x.Company == company).OrderByDescending(y => y.Age).First());
        }
        return result;
    }
    public static void Main()
```

Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

- Test case 0
- Test case 1
- Test case 2
- Test case 3
- Test case 4
- Test case 5
- Test case 6

Input (stdin) — Run as Custom Input | Download

```
12
Ainslee Ginsie Galaxy 28
Libbey Apdell Starbucks 44
Illa Stebbings Berkshire 49
Laina Sycamore Berkshire 20
Abbe Parnell Amazon 20
Ludovika Reveley Berkshire 30
Rene Antos Galaxy 44
Vinson Beckenham Berkshire 45
Reed Lynock Amazon 41
Wyndham Bamfield Berkshire 34
Loraine Sappson Amazon 49
```

Test case result list

Sample Input For Custom Testing

```
12
Sybila Fulle Kimberly 24
Scarface Stork Tesla 22
Ashli Crosseland Kimberly 36
Allene Stebbings Galaxy 19
Valentin Harbert Amazon 28
Gracie Pappin Tesla 44
Sadye Orcott Rockwell 30
Timoteo Pook Amazon 35
Marris Apdell Rockwell 43
Pen Ghilardini Rockwell 38
Bern Aizikov Rockwell 20
Sela Farrier Amazon 47
```

Sample Output

```
The average age for company Amazon is 37
The average age for company Galaxy is 19
The average age for company Kimberly is 30
The average age for company Rockwell is 33
The average age for company Tesla is 33
The count of employees for company Amazon is 3
The count of employees for company Galaxy is 1
The count of employees for company Kimberly is 2
The count of employees for company Rockwell is 4
The count of employees for company Tesla is 2
The oldest employee of company Amazon is Sela Farrier having age 47
The oldest employee of company Galaxy is Allene Stebbings having age 19
The oldest employee of company Kimberly is Ashli Crosseland having age 36
The oldest employee of company Rockwell is Marris Apdell having age 43
The oldest employee of company Tesla is Gracie Pappin having age 44
```

Explanation

Here, 12 employees are presented with their details (first name, last name, company name, and age). Then, based on that data, the 3 methods are called: *AverageAgeForEachCompany*, *CountOfEmployeesForEachCompany*, and *OldestAgeForEachCompany*. The result obtained from these 3 function calls is printed to the standard output.

Language: C# | Environment | Autocomplete Ready

```
            {
                result.Add(company, (int)employees.Where(x => x.Company == company).Count());
            }
            return result;
        }

        public static Dictionary<string, Employee> OldestAgeForEachCompany(List<Employee> employees)
        {
            var result = new Dictionary<string, Employee>();
            foreach(var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
            {
                result.Add(company, employees.Where(x => x.Company == company).OrderByDescending(y => y.Age).First());
            }
            return result;
        }
        public static void Main()…
```

Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Input (stdin) — Run as Custom Input | Download

```
12
Sybila Fulle Kimberly 24
Scarface Stork Tesla 22
Ashli Crosseland Kimberly 36
Allene Stebbings Galaxy 19
Valentin Harbert Amazon 28
Gracie Pappin Tesla 44
Sadye Orcott Rockwell 30
Timoteo Pook Amazon 35
Marris Apdell Rockwell 43
Pen Ghilardini Rockwell 38
Bern Aizikov Rockwell 20
```

```
The count of employees for company Berkshire is 5
The count of employees for company Galaxy is 2
The count of employees for company Starbucks is 2
The oldest employee of company Amazon is Loraine Sappson having age 49
The oldest employee of company Berkshire is Illa Stebbings having age 49
The oldest employee of company Galaxy is Rene Antos having age 44
The oldest employee of company Starbucks is Abbe Antonutti having age 47
```

Explanation

Here, 12 employees are presented with their details (first name, last name, company name, and age). Then, based on that data, the 3 methods are called: *AverageAgeForEachCompany*, *CountOfEmployeesForEachCompany*, and *OldestAgeForEachCompany*. The result obtained from these 3 function calls is printed to the standard output.

▾ Sample Case 1

Sample Input For Custom Testing

```
12
Sybila Fulle Kimberly 24
Scarface Stork Tesla 22
Ashli Crosseland Kimberly 36
Allene Stebbings Galaxy 19
Valentin Harbert Amazon 28
Gracie Pappin Tesla 44
Sadye Orcott Rockwell 30
Timoteo Pook Amazon 35
Marris Apdell Rockwell 43
Pen Ghilardini Rockwell 38
Bern Aizikov Rockwell 20
Sela Farrier Amazon 47
```

Sample Output

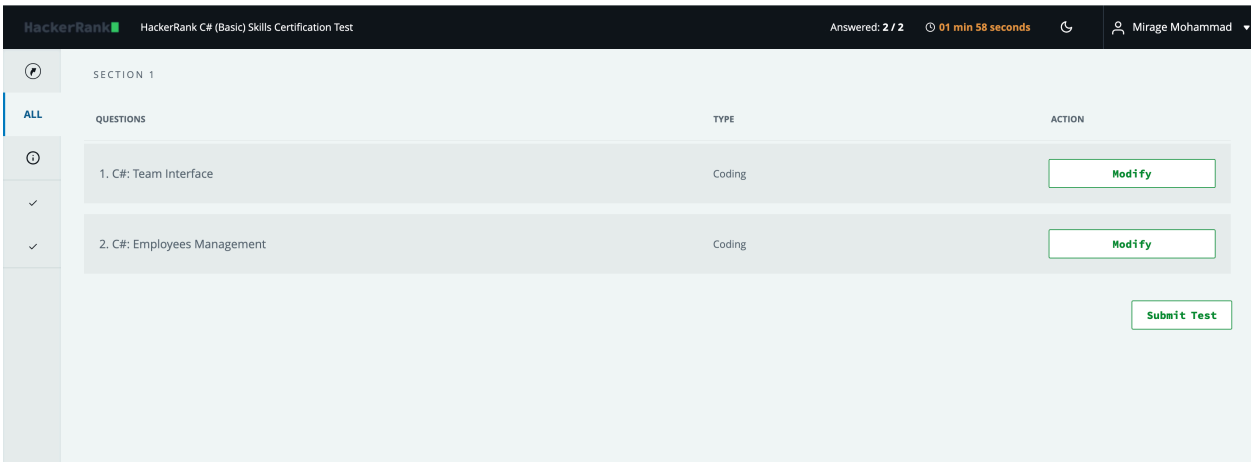
```
The average age for company Amazon is 37
The average age for company Galaxy is 19
The average age for company Kimberly is 30
The average age for company Rockwell is 33
The average age for company Tesla is 33
The count of employees for company Amazon is 3
The count of employees for company Galaxy is 1
The count of employees for company Kimberly is
```

Language: C# | Environment | Autocomplete Ready

```
using System;…
        public static Dictionary<string, int> AverageAgeForEachCompany(List<Employee> employees)
        {
            var result = new Dictionary<string, int>();
            foreach( var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
            {
                result.Add(company, (int)Math.Round(employees.Where(x => x.Company == company).Average(y => y.Age), 0)
);
            }
            return result;

        }

        public static Dictionary<string, int> CountOfEmployeesForEachCompany(List<Employee> employees)
        {
            var result = new Dictionary<string, int>();
            foreach(var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
            {
                result.Add(company, (int)employees.Where(x => x.Company == company).Count());
            }
            return result;
        }

        public static Dictionary<string, Employee> OldestAgeForEachCompany(List<Employee> employees)
        {
            var result = new Dictionary<string, Employee>();
            foreach(var company in employees.Select(x => x.Company).Distinct().OrderBy(x => x))
            {
                result.Add(company, employees.Where(x => x.Company == company).OrderByDescending(y => y.Age).First());
            }
            return result;
        }
        public static void Main()…
```

Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit


HackerRank
HackerRank C# (Basic) Skills Certification Test
Answered: 2 / 2
01 min 58 seconds
Mirage Mohammad
SECTION 1
ALL
QUESTIONS
TYPE
ACTION
1. C#: Team Interface
Coding
Modify
2. C#: Employees Management
Coding
Modify
Submit Test