# Exercise: Objects and Classes

Problems for exercise and homework for the ["C# Fundamentals" course @ SoftUni](https://softuni.bg/trainings/3836/programming-fundamentals-with-csharp-september-2022)  
You can check your solutions in [Judge](https://judge.softuni.org/Contests/1215/Objects-and-Classes-Exercise)

## Advertisement Message

Create a program that **generates random fake advertisement messages** to advertise a product. The messages must consist of 4 parts: **phrase** + **event** + **author** + **city**. Use the following predefined parts:

* **Phrases** – {"Excellent product.", "Such a great product.", "I always use that product.", "Best product of its category.", "Exceptional product.", "I can't live without this product."}
* **Events** – {"Now I feel good.", "I have succeeded with this product.", "Makes miracles. I am happy of the results!", "I cannot believe but now I feel awesome.", "Try it yourself, I am very satisfied.", "I feel great!"}
* **Authors** – {"Diana", "Petya", "Stella", "Elena", "Katya", "Iva", "Annie", "Eva"}
* **Cities** – {"Burgas", "Sofia", "Plovdiv", "Varna", "Ruse"}

The format of the output message is the following: "**{phrase} {event} {author}** – **{city}**."

You will receive the **number of messages** to be generated. Print each random message at a separate line.

using System;

namespace \_01.\_Advertisement\_Message

{

class Program

{

static void Main(string[] args)

{

Random random = new Random();

string[] phrases = new string[6] { "Excellent product.", "Such a great product.", "I always use that product.", "Best product of its category.", "Exceptional product.", "I can't live without this product." };

string[] events = new string[6] { "Now I feel good.", "I have succeeded with this product.", "Makes miracles. I am happy of the results!", "I cannot believe but now I feel awesome.", "Try it yourself, I am very satisfied.", "I feel great!" };

string[] names = new string[8] { "Diana", "Petya", "Stella", "Elena", "Katya", "Iva", "Annie", "Eva" };

string[] cities = new string[5] { "Burgas", "Sofia", "Plovdiv", "Varna", "Ruse" };

int n = int.Parse(Console.ReadLine());

for (int i = 0; i < n; i++)

{

int ind1 = random.Next(0, 6);

int ind2 = random.Next(0, 6);

int ind3 = random.Next(0, 8);

int ind4 = random.Next(0, 5);

Console.WriteLine($"{phrases[ind1]} {events[ind2]} {names[ind3]} – {cities[ind4]}.");

}

}

}

}

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 | Such a great product. Now I feel good. Elena – Ruse.  Excellent product. Makes miracles. I am happy of the results! Katya – Varna.  Best product of its category. That makes miracles. Eva – Sofia. |

## Articles

Create a **class** **Article** with the following properties:

* **Title** – a string
* **Content** – a string
* **Author** – a string

The class should have a constructor and the following methods:

* **Edit (new content**) – change the old content with the new one
* **ChangeAuthor (new author)** – change the author
* **Rename (new title)** – change the title of the article
* Override the **ToString** method – print the article in the following format:

**"{title} - {content}: {author}"**

Create a program that reads an article in the following format **"{title}, {content}, {author}"**. On the next line, you will receive a number **n,** representing the number of commands, which will follow after it. On the next **n lines,** you will be receiving the following commands:

* **"Edit: {new content}"**
* **"ChangeAuthor: {new author}"**
* **"Rename: {new title}"**

In the end, print the final state of the article.

using System;

using System.Linq;

namespace \_02.\_Articles

{

class Program

{

static void Main(string[] args)

{

string[]input = Console.ReadLine()

.Split(", ")

.ToArray();

Article article = new Article(input[0], input[1], input[2]);

int n = int.Parse(Console.ReadLine());

for (int i = 0; i < n; i++)

{

string[] cmdArgs = Console.ReadLine()

.Split(": ")

.ToArray();

string command = cmdArgs[0];

if (command=="Edit")

{

article.Content = cmdArgs[1];

}

else if (command == "ChangeAuthor")

{

article.Author = cmdArgs[1];

}

else if (command == "Rename")

{

article.Title = cmdArgs[1];

}

}

Console.WriteLine(article.ToString());

}

}

public class Article

{

public Article(string title, string content, string author)

{

Title = title;

Content = content;

Author = author;

}

public string Title { get; set; }

public string Content { get; set; }

public string Author { get; set; }

public override string ToString()

{

return $"{ this.Title} - { this.Content}: {this.Author}";

}

}

}

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| some title, some content, some author  3  Edit: better content  ChangeAuthor: better author  Rename: better title | better title - better content: better author |
| Fight club, love story, Martin Scorsese  2  Edit: underground fight club that evolves into much more  ChangeAuthor: Chuck Palahniuk | Fight club - underground fight club that evolves into much more: Chuck Palahniuk |

## Articles 2.0

Change the program from the previous problem in such a way, that you will be able to store a **list of articles**. You will not need to use the previous methods anymore (**except the** "**ToString()**"). On the **first line,** you will receive the number of articles. On the **next lines,** you will receive the **articles in the same format** as in the previous problem: **"{title}, {content}, {author}"**. Print the articles.

using System;

using System.Collections.Generic;

using System.Linq;

namespace \_03.\_Articles\_2.\_0

{

class Program

{

static void Main(string[] args)

{

List<Article> articles = new List<Article>();

int n = int.Parse(Console.ReadLine());

for (int i = 0; i < n; i++)

{

string[] input = Console.ReadLine()

.Split(", ")

.ToArray();

Article article = new Article(input[0], input[1], input[2]);

articles.Add(article);

}

for (int i = 0; i < articles.Count; i++)

{

Console.WriteLine(articles[i].ToString());

}

}

}

public class Article

{

public Article(string title, string content, string author)

{

Title = title;

Content = content;

Author = author;

}

public string Title { get; set; }

public string Content { get; set; }

public string Author { get; set; }

public override string ToString()

{

return $"{ this.Title} - { this.Content}: {this.Author}";

}

}

}

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  Science, planets, Bill  Article, content, Johnny  title | Science - planets: Bill Article - content: Johnny |
| 3  title1, C, author1  title2, B, author2  title3, A, author3  content | title1 - C: author1  title2 - B: author2  title3 - A: author3 |

## Students

Create a program that sorts some students by their grade in descending order. Each student should have:

* **First name** (String)
* **Last name** (String)
* **Grade** (a floating-point number)

### Input

* On the first line, you will receive a number **n** - the **count of all students.**
* On the next **n** lines, you will be receiving information about these students in the following format: **"{first name} {second name} {grade}".**

### Output

* Print out the information about each student in the following format: **"{first name} {second name}: {grade}".**

using System;

using System.Linq;

using System.Collections.Generic;

namespace \_04.\_Students

{

class Program

{

static void Main(string[] args)

{

int n = int.Parse(Console.ReadLine());

List<Student> students = new List<Student>();

for (int i = 0; i < n; i++)

{

string[] cmdArgs = Console.ReadLine()

.Split();

string firstName = cmdArgs[0];

string lastName = cmdArgs[1];

double grade = double.Parse(cmdArgs[2]);

Student student = new Student(firstName, lastName, grade);

students.Add(student);

}

students = students.OrderByDescending(x => x.Grade).ToList();

foreach (var student in students)

{

Console.WriteLine($"{student.FirstName} { student.LastName}: {student.Grade:f2}");

}

}

}

public class Student

{

public Student(string firstName, string lastName, double grade)

{

FirstName = firstName;

LastName = lastName;

this.Grade = grade;

}

public string FirstName { get; set; }

public string LastName { get; set; }

public double Grade { get; set; }

}

}

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Lakia Eason 3,90  Prince Messing 5,49  Akiko Segers 4,85  Rocco Erben 6,00 | Rocco Erben: 6.00  Prince Messing: 5.49  Akiko Segers: 4.85  Lakia Eason: 3.90 |
| 3  Mary Elizabeth 4.22  Li Xiao 5.74  Liz Smith 4.87 | Li Xiao: 5.74  Liz Smith: 4.87  Mary Elizabeth: 4.22 |

## Teamwork Projects

It's time for the teamwork projects and you are responsible for gathering the teams. First, you will receive an integer – the **count** of the **teams** you will have to **register**. You will be given a **user** and a **team**, separated with "-". The user is the **creator** of **the team**. For every newly created team you should **print** a message:

"Team {teamName} has been created by {user}!".

Next, you will receive а user with a team, separated with **"->"**, which means that the user wants to **join** that **team**. Upon receiving the command: "end of assignment", you should print **every team**, **ordered** by the **count** of its **members** (**descending**) and then by **name** (**ascending**). For each team, you have to print its members **sorted** by name (**ascending**). However, there are several **rules**:

* If а user tries to **create** a team more than once, a message should be displayed:
  + "Team {teamName} was already created!"
* A creator of a team **cannot** **create** another team – the following message should be thrown:
  + "{user} cannot create another team!"
* If а user tries to **join** a non-existent team, a message should be displayed:
  + "Team {teamName} does not exist!*"*
* A member of a team **cannot** **join** another team – the following message should be thrown:
  + "Member {user} cannot join team {team Name}!"
* In the end,teams with **zero** members (with **only a creator**) should **disband** and you have toprint them **ordered by name in ascending order**.
* Every **valid** team should be printed ordered by **name** (ascending) in the following format:

|  |
| --- |
| "{teamName}  - {creator}  -- {member}…" |
|  |

using System;

using System.Collections.Generic;

using System.Linq;

namespace \_05.\_Teamwork\_Projects

{

class Program

{

static void Main(string[] args)

{

List<Team> teams = new List<Team>();

InitializeTeam(teams);

NewMethod(teams);

PrintingTeamsWithMembers(teams);

PrintteamsWithoutMembers(teams);

}

private static void PrintteamsWithoutMembers(List<Team> teams)

{

List<Team> teamsWithouthMembers = teams

.Where(t => t.Members.Count == 0)

.ToList();

Console.WriteLine("Teams to disband:");

foreach (var team in teamsWithouthMembers.OrderBy(x => x.Name))

{

Console.WriteLine($"{team.Name}");

}

}

private static void PrintingTeamsWithMembers(List<Team> teams)

{

List<Team> teamsWithMembers = teams

.Where(t => t.Members.Count > 0)

.OrderByDescending(t => t.Members.Count)

.ThenBy(t => t.Name)

.ToList();

foreach (var team in teamsWithMembers)

{

Console.WriteLine($"{team.Name}");

Console.WriteLine($"- {team.Creator}");

foreach (string memberName in team.Members.OrderBy(x => x))

{

Console.WriteLine($"-- { memberName}");

}

}

}

private static void NewMethod(List<Team> teams)

{

string input = Console.ReadLine();

while (input != "end of assignment")

{

string[] cmdArgs = input.Split("->");

string user = cmdArgs[0];

string team = cmdArgs[1];

if (!teams.Any(x => x.Name == team))

{ Console.WriteLine($"Team {team} does not exist!"); }

else if (teams.Any(x => x.Creator == user) || teams.Any(x => x.Members.Contains(user)))

{ Console.WriteLine($"Member {user} cannot join team {team}!"); }

else

{

Team teamsToJoin = teams.First(t => t.Name == team);

teamsToJoin.AddMember(user);

}

input = Console.ReadLine();

}

}

private static void InitializeTeam(List<Team> teams)

{

int n = int.Parse(Console.ReadLine());

for (int i = 0; i < n; i++)

{

string[] cmdArg = Console.ReadLine().Split('-');

string creator = cmdArg[0];

string teamName = cmdArg[1];

if (teams.Any(x => x.Name == teamName))

{ Console.WriteLine($"Team {teamName} was already created!"); }

else if (teams.Any(x => x.Creator == creator))

{ Console.WriteLine($"{creator} cannot create another team!"); }

else

{

Team team = new Team(teamName, creator);

teams.Add(team);

Console.WriteLine($"Team {teamName} has been created by {creator}!");

}

}

}

}

public class Team

{

private readonly List<string> members;

public Team(string name, string creator)

{

Name = name;

Creator = creator;

this.members = new List<string>();

}

public string Name { get; set; }

public string Creator { get; set; }

public List<string> Members

=> members;

public void AddMember(string membername)

{

this.members.Add(membername);

}

}

}

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  John-PowerPuffsCoders  Tony-Tony is the best  Peter->PowerPuffsCoders  Tony->Tony is the best  end of assignment | Team PowerPuffsCoders has been created by John!  Team Tony is the best has been created by Tony!  Member Tony cannot join team Tony is the best!  PowerPuffsCoders  - John  -- Peter  Teams to disband:  Tony is the best | Tony created a team, which he attempted to join later and this action resulted in throwing a certain message. Since nobody else tried to join his team, the team had to **disband**. |
| 3  Tanya-CloneClub  Helena-CloneClub  Tedy-SoftUni  George->softUni  George->SoftUni  Tatyana->Leda  John->SoftUni  Cossima->CloneClub  end of assignment | Team CloneClub has been created by Tanya!  Team CloneClub was already created!  Team SoftUni has been created by Tedy!  Team softUni does not exist!  Team Leda does not exist!  SoftUni  - Tedy  -- George  -- John  CloneClub  - Tanya  -- Cossima  Teams to disband: | Note that when a user joins a team, you should first check if the team exists and then check if the user is already in a team:  Tanya has created CloneClub, then she tried to join a non-existent team and the concrete message was displayed. |

## Vehicle Catalogue

Until you receive the "**End**" command, you will be receiving lines of input in the following format:

|  |
| --- |
| "{typeOfVehicle} {model} {color} {horsepower}" |

When you receive the "**End**" command, you will start receiving information about some **vehicles**.

**For every vehicle**, print out the information about it in the following **format**:

|  |
| --- |
| "Type: {typeOfVehicle}  Model: {modelOfVehicle}  Color: {colorOfVehicle}  Horsepower: {horsepowerOfVehicle} |

When you receive the "Close the Catalogue" command, print out the average horsepower of the cars and the average horsepower of the trucks in the format:

"{typeOfVehicles} have average horsepower of {averageHorsepower}."

The average horsepower is calculated by **dividing the sum of the horsepower of all vehicles of the given type by the total count of all vehicles from that type.** Format the answer to the **second digit after the decimal point.**

using System;

using System.Collections.Generic;

using System.Linq;

namespace \_06.\_Vehicle\_Catalogue

{

class Program

{

static void Main(string[] args)

{

string command = Console.ReadLine();

List<Vehicle> catalog = new List<Vehicle>();

while(command!="End")

{

string[] cmdArgs = command.Split(" ");

string vehicleType = cmdArgs[0];

string model = cmdArgs[1];

string color = cmdArgs[2];

int horsePower = int.Parse(cmdArgs[3]);

Vehicle vehicle = new Vehicle(vehicleType, model, color, horsePower);

catalog.Add(vehicle);

command = Console.ReadLine();

}

command = Console.ReadLine();

while(command!= "Close the Catalogue")

{

List<Vehicle> searchedVehicles = catalog

.Where(x => x.Model == command)

.ToList();

foreach (var vehicle in searchedVehicles)

{

if(vehicle.VehicleType=="car")

{ Console.WriteLine($"Type: Car"); }

else

{ Console.WriteLine($"Type: Truck"); }

Console.WriteLine($"Model: {vehicle.Model}");

Console.WriteLine($"Color: {vehicle.Color}");

Console.WriteLine($"Horsepower: {vehicle.HorsePower}");

}

command = Console.ReadLine();

}

List<Vehicle> cars = catalog.Where(x => x.VehicleType == "car").ToList();

int sum=0;

if (cars.Count > 0)

{

foreach (var item in cars)

{

sum += item.HorsePower;

}

Console.WriteLine($"Cars have average horsepower of: {((double)sum / cars.Count):f2}.");

}

else

{

Console.WriteLine($"Cars have average horsepower of: {0:f2}.");

}

List<Vehicle> trucks = catalog.Where(x => x.VehicleType == "truck").ToList();

sum = 0;

if (trucks.Count > 0)

{

foreach (var item in trucks)

{

sum += item.HorsePower;

}

Console.WriteLine($"Trucks have average horsepower of: {((double)sum / trucks.Count):f2}.");

}

else

{

Console.WriteLine($"Trucks have average horsepower of: {0:f2}.");

}

}

}

public class Vehicle

{

public Vehicle(string vehicleType, string model, string color, int horsePower)

{

VehicleType = vehicleType;

Model = model;

Color = color;

HorsePower = horsePower;

}

public string VehicleType { get; set; }

public string Model { get; set; }

public string Color { get; set; }

public int HorsePower { get; set; }

}

}

### Constraints

* The type of vehicle will always be either a **car** or a **truck**.
* You will not receive the **same** **model** **twice**.
* The received horsepower will be an integer in the range **[1…1000].**
* You will receive at most **50** vehicles.
* The separator will always be single **whitespace**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| truck Man red 200  truck Mercedes blue 300  car Ford green 120  car Ferrari red 550  car Lamborghini orange 570  End  Ferrari  Ford  Man  Close the Catalogue | Type: Car  Model: Ferrari  Color: red  Horsepower: 550  Type: Car  Model: Ford  Color: green  Horsepower: 120  Type: Truck  Model: Man  Color: red  Horsepower: 200  Cars have average horsepower of: 413.33.  Trucks have average horsepower of: 250.00. |
| truck Volvo blue 220  truck Man red 350  car Tesla silver 450  car Nio red 650  truck Mack white 430  car Koenigsegg orange 750  End  Tesla  Nio  Man  Mack  Close the Catalogue | Type: Car  Model: Tesla  Color: silver  Horsepower: 450  Type: Car  Model: Nio  Color: red  Horsepower: 650  Type: Truck  Model: Man  Color: red  Horsepower: 350  Type: Truck  Model: Mack  Color: white  Horsepower: 430  Cars have average horsepower of: 616.67.  Trucks have average horsepower of: 333.33. |

## Order by Age

You will receive **an unknown number of lines**. On each line you will receive an **array with 3 elements**:

* The **first** element is a **string** - the **name of the person**
* The **second** element a **string** - the **ID of the person**
* The **third** element is an **integer** - the **age of the person**

If you get a person whose **ID you have already received before**, **update** the name and age for that ID with that **of the new person**. When you receive the command "**End**", print **all of the** **people**, **ordered** by **age**.

using System;

using System.Collections.Generic;

using System.Linq;

namespace \_07.\_Order\_by\_Age

{

class Program

{

static void Main(string[] args)

{

string input = Console.ReadLine();

List<People> people = new List<People>();

while(input!="End")

{

string[] cmdArgs = input.Split(" ");

string name = cmdArgs[0];

int Id = int.Parse(cmdArgs[1]);

int age= int.Parse(cmdArgs[2]);

People person = new People(name, Id, age);

people.Add(person);

input = Console.ReadLine();

}

people = people.OrderBy(x => x.Age).ToList();

foreach (var person in people)

{

Console.WriteLine($"{person.Name} with ID: {person.ID} is {person.Age} years old.");

}

}

}

public class People

{

public People(string name, int iD, int age)

{

Name = name;

ID = iD;

Age = age;

}

public string Name { get; set; }

public int ID{ get; set; }

public int Age { get; set; }

}

}

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| George 123456 20  Peter 78911 15  Stephen 524244 10  End | Stephen with ID: 524244 is 10 years old.  Peter with ID: 78911 is 15 years old.  George with ID: 123456 is 20 years old. |
| Lewis 123456 20  James 78911 15  Robert 523444 11  Jennifer 345244 13  Mary 52424678 22  Patricia 567343 54  End | Robert with ID: 523444 is 11 years old.  Jennifer with ID: 345244 is 13 years old.  James with ID: 78911 is 15 years old.  Lewis with ID: 123456 is 20 years old.  Mary with ID: 52424678 is 22 years old.  Patricia with ID: 567343 is 54 years old. |