# Exercise: Objects and Classes

You can check your solutions in [Judge.](https://alpha.judge.softuni.org/contests/objects-and-classes-exercise/1327)

## Advertisement Message

Write a program that **generates random fake advertisement messages** to extol some product. The messages must consist of 4 parts: **laudatory** **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: "**{phrase} {event} {author} – {city}**".

As an input, you take the **number of messages** to be generated. Print each random message on a separate line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 | Such a great product. Now I feel good. Elena – Ruse  Excelent product. Makes miracles. I am happy of the results! Katya – Varna  Best product of its category. That makes miracles. Eva – Sofia |
| 4 | I always use that product. Makes miracles. I am happy of the results! Iva - Ruse  I can’t live without this product. I cannot believe but now I feel awesome. Katya - Burgas  Such a great product. Try it yourself, I am very satisfied. Iva - Varna  Best product of its category. I cannot believe but now I feel awesome. Eva - Ruse |

## Articles

Create an article class 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 **ToString** – print the article in the following format:

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

Write a program that reads an article in the following format **"{title}, {content}, {author}"**. On the next line, you will get the number **n**. On the next **n lines,** you will get one of the following commands:

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

At the end, print the final article.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| some title, some content, some author  3  Edit: better content  ChangeAuthor: better author  Rename: better title | better title - better content: better author |
| Holy Ghost, content, John Sandford  3  ChangeAuthor: Mitch Albom  ChangeAuthor: better author  ChangeAuthor: Kim Heacox | Holy Ghost - content: Kim Heacox |

## Opinion Poll

Using the Person class, write a program that reads from the console **N** lines of personal information and then prints all people whose **age** is **more than 30** years.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Peter 12  Sam 31  Itan 48 | Sam – 31 Itan - 48 |
| 5  Niko 33  Yana 88  Todor 22  Lisa 44  Sam 11 | Niko - 33  Yana - 88  Lisa - 44 |

## Students

Write a program that receives **n count of students** and **orders them by grade** (in **descending**). Each student should have a **first name** (string), a **last name** (string), and a **grade** (a floating-point number).

### Input

* First-line will be a number **n.**
* Next **n** lines you will get student info in the format **"{first name} {second name} {grade}".**

### Output

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

### 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 |
| 4  Sydnie Britton 5.79  Amias Mathews 2.30  Mora Tod 2.78  Pete Kendrick 2.61 | Sydnie Britton: 5.79  Mora Tod: 2.78  Pete Kendrick: 2.61  Amias Mathews: 2.30 |

## Vehicle Catalogue

You have to make a catalog for vehicles. You will receive two types of vehicles - a **car** or a **truck**.

Until you receive the command "**End**" you will receive **lines** of **input** in the format:

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

After the "**End**" command, you will start receiving **models** of **vehicles**. Print for every received vehicle its **data** in the format:

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

When you receive the command "**Close the Catalogue**", stop receiving input and print the **average** **horsepower** for the **cars** and the **trucks** in the format:

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

The **average** **horsepower** is calculated by **dividing** the **sum** of **horsepower** for **all** vehicles of the type by the **total** **count** of **vehicles** from the **same** **type**.

Format the answer to the **2nd decimal point**.

### Constraints

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

### 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. |
| car Opel green 736  End  Close the Catalogue | Cars have average horsepower of: 736.00.  Trucks have average horsepower of: 0.00. |

## 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 will be a **string** and represents the **name** of the person. **The second** element will be a **string** and will represent the **ID** of the person. **The last** element will be an **integer** which represents the **age** of the person.

When you receive the command "**End**", stop taking input and print **all the** **people**, **ordered** by **age**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| George 123456 20  Peter 78911 15  Stephan 524244 10  End | Stephan with ID: 524244 is 10 years old.  Peter with ID: 78911 is 15 years old.  George with ID: 123456 is 20 years old. |
| Cindy 88611 2  Kaloyan 13967 3  Simona 53316 11  Gil 31837 72  End | Cindy with ID: 88611 is 2 years old.  Kaloyan with ID: 13967 is 3 years old.  Simona with ID: 53316 is 11 years old.  Gil with ID: 31837 is 72 years old. |

## \*Grooming Salon

### Preparation

Download the skeleton provided in Judge. **Do not** change the **packages**!

**Pay attention to the name of the package groomingSalon, all the classes, their fields, and methods the same way they are presented in the following document. It is also important to keep the project structure as described.**

## Problem description

Your task is to create a repository, which stores items by creating the classes described below.

First, write a Java class **Pet** with the following fields:

* **name: String**
* **age: int**
* **owner: String**

The class **constructor** should receive **all fields.** You need to create the appropriate **getters and setters**. The class should override the **toString()** method in the following format:

**"{name} {age} - ({owner})"**

**Next**, write a Java class **GroomingSalon** that has **data** (a collection, which stores the Pets). All entities inside the repository have the **same fields**. Also, the **GroomingSalon** class should have those fields:

* **capacity: int**

The class **constructor** should receive **capacity**, also it should initialize the **data** with a new instance of the collection**.** Implement the following features:

* Field **data** – **List** that holds added pets
* Method add(Pet pet) – **adds** an **entity** to the data **if** **there** **is** an **empty place** inthegrooming salon for the pet.
* Method remove(String name) – removes the pet by **given name,** if such **exists**, and **returns boolean**.
* Method **getPet(String name, String owner)** – returns the pet with the **given name** and **owner** or **null if no such pet exists**.
* Getter getCount – **returns** the **number** of pets.
* **getStatistics()** – **returns** a **String** in the following **format**:
  + **"** **The grooming salon has the following clients:  
    {name} {owner}  
    {name} {owner}**

**(…)**"

## Constraints

* The **combinations** of **names** and **owners** will **always be unique**.
* The **age** of the pets will always be **positive**.

## Examples

This is an example of how the **GroomingSalon** class is **intended to be used**.

|  |
| --- |
| Sample code usage |
| // Initialize the repository  GroomingSalon salon = new GroomingSalon(20);  // Initialize entity  Pet dog = new Pet("Ellias", 5, "Tim");  // Print Pet  System.out.println(dog); // Ellias 5 - (Tim)  // Add Pet  salon.add(dog);  // Remove Pet  System.out.println(salon.remove("Ellias")); // true  System.out.println(salon.remove("Pufa")); // false  Pet cat = new Pet("Bella", 2, "Mia");  Pet bunny = new Pet("Zak", 4, "Jon");  salon.add(cat);  salon.add(bunny);  // Get Pet  Pet pet = salon.getPet("Bella", "Mia");  System.out.println(pet); // Bella 2 - (Mia)  // Count  System.out.println(salon.getCount()); // 2  // Get Statistics  System.out.println(salon.getStatistics());  // The grooming salon has the following clients:  //Bella Mia  //Zak Jon |

## Submission

Zip all the files in the project folder except the **bin** and **obj** folders.

Submit a **single .zip file**, containing **groomingSalon package, with the classes inside (Pet, GroomingSalon, and the Main class)**, there is no specific content required inside the Main class e. g. you can do any kind of local testing of your program there. However, there should be a **main(String[] args)** method inside.