# Problem 3. Veterinary Clinic

class VeterinaryClinic {  
 *// TODO: implement this class...*  
}

Your Task

Write a class VeterinaryClinic, which implements the following functionality:

Functionality

#### constructor (clinicName, capacity)

Receives 2 parameters at initialization of the class (clinicName and capacity).

Should have these **3** properties:

* **clinicName** – property of type string;
* **capacity** – property of type number;
* **clients** – initially an empty array;

**Hint:** Add more properties like **totalProfit** and **currentWorkload** to help you finish the task. Read the problem description **until the end** to get clear with the requirements.

#### newCustomer(ownerName, petName, kind, procedures)

The ownerName, petName and kind are of type string and the procedures are an array of strings. This information will be used in the following toString() method.

**ownerName** – string that keeps the name of the current pet owner, one owner may have more than one pets under his name, choose customer structure wisely to collect all of the given information. Once stored this information stays in the clinic data, even when the pet is healed.

#### petName – string that keeps the name of the current pet, once stored this information stays in the clinic data, even when the pet is healed.

#### kind – string that keeps the current pet kind, be careful of upper cases into the input string. Once stored this information stays in the clinic data, even when the pet is healed.

#### procedures – array of strings that keeps the procedures the current pet kind needs. You know that a pet is a current client when one or more procedures are recorded at his list of procedures. When pet is healed and leaves the clinic the array of procedures must be emptied. So when the pet comes back again for healing it can be listed with new procedures.

Before register:

* Check if the clinic is able to accept more pets. If the clinic is full throw an Error:

"Sorry, we are not able to accept more patients!"

* Check if the pet is already registered under this client name. If it's registered and still has full list of procedures, you should throw an Error:

"This pet is already registered under { ownerName } name! { petName } is on our lists, waiting for { all his procedures separated by ', ' }."

* Otherwise this function should add the customer and his pet, update the current clinic workload and return:

"Welcome { petName }!"

#### onLeaving (ownerName, petName)

* Check if the given **ownerName** corresponds to a client in the **clients** array, if not **throw an Error**:

"Sorry, there is no such client!"

* Then check if the given petName is registered under this ownerName, if not or it is registered but the procedures array is empty because all his procederues are done , then throw an Error:

"Sorry, there are no procedures for { petName }!"

* Otherwise, on leaving you have to **collect** the current client bill, add it to the total **clinic profit** and **save the data**. Calculate the bill knowing that each **procedure** cost **500.00$.** Do not forget to **update** thecurrent **workload**. Clear the array with **procedures** for the current pet.

When pet leaves the clinic, **petName** and **kind** should stay like information in our data, but with **no more** procedures in the **array of procedures**. After that **return,** the following string:

**"Goodbye { petName }. Stay safe!"**

#### toString ()

Return the full information of the clinic.

* First, we have to **calculate** how busy the clinic is in **percentages**. Percentage represents all current **pets** that have **procedures** based on the **full capacity** of the clinic. The percentage is rounded to the nearest smaller number:

"{ clinicName } is { percentage }% busy today!"

* On the second line comes the collected profit, that must be fixed to the second digit after the decimal point:

"Total profit: { profit }$"

On the next lines, return the whole information for the owners in the following format**.** Print **kind** property with **lowercase** letters. All owners should be in **alphabetical** **order**, as also pets of each of them must be in **alphabetical** **order** too:

**"{ ownerName } with:**

**---{ petName } - a { kind } that needs: { procedures separated by ', '}"**

Examples

This is an example how the code is intended to be used:

|  |
| --- |
| Sample code usage |
| let clinic = new VeterinaryClinic('SoftCare', 10);  console.log(clinic.newCustomer('Jim Jones', 'Tom', 'Cat', ['A154B', '2C32B', '12CDB']));  console.log(clinic.newCustomer('Anna Morgan', 'Max', 'Dog', ['SK456', 'DFG45', 'KS456']));  console.log(clinic.newCustomer('Jim Jones', 'Tiny', 'Cat', ['A154B']));  console.log(clinic.onLeaving('Jim Jones', 'Tiny'));  console.log(clinic.toString());  clinic.newCustomer('Jim Jones', 'Sara', 'Dog', ['A154B']);  console.log(clinic.toString()); |
| Corresponding output |
| **Welcome Tom!**  **Welcome Max!**  **Welcome Tiny!**  **Goodbye Tiny. Stay safe!**  **SoftCare is 20% busy today!**  **Total profit: 500.00$**  **Anna Morgan with:**  **---Max - a dog that needs: SK456, DFG45, KS456**  **Jim Jones with:**  **---Tiny - a cat that needs:**  ---Tom - a cat that needs: A154B, 2C32B, 12CDB  SoftCare is 30% busy today!  Total profit: 500.00$  Anna Morgan with:  ---Max - a dog that needs: SK456, DFG45, KS456  Jim Jones with:  ---Sara - a dog that needs: A154B  ---Tiny - a cat that needs:  ---Tom - a cat that needs: A154B, 2C32B, 12CDB |

*GOOD LUCK!😊*