**Problem 3. Parking**

Exam problems for the [“JavaScript Advanced” course @ SoftUni](https://softuni.bg/courses/javascript-advanced). Submit your solutions in the SoftUni Judge system at <https://judge.softuni.bg/Contests/Compete/Index/2590#2>

Write a **class Parking**, which implements the following functionality:

**Functionality**

**constructor ( capacity )**

Should have these **2** properties:

* **capacity** – **number**;
* **vehicles** – **array**;

**Hint:** You can add more properties to help you finish the task.

**addCar( carModel, carNumber )**

The **carModel** and **carNumber** are of type **string**.

* If there's **not enough parking spots** for the car the park, **throw an Error**:

**"Not enough parking space."**

* Otherwise this function should **add** the car, with the properties: **carModel**, **carNumber**, **payed**: **default false**, to the vehicles arrayand **return:**

**"The {carModel}, with a registration number {carNumber}, parked."**

**removeCar( carNumber )**

* If the car is not found, throw an Error:

**"The car, you're looking for, is not found."**

* If the car hasn't payed, throw an Error:

**"{carNumber} needs to pay before leaving the parking lot."**

* Otherwise, this function should **remove** the car from the vehicles arrayand **return:**

**"{carNumber} left the parking lot."**

**pay( carNumber )**

* If the car is not found, throw an Error:

**"{carNumber} is not in the parking lot."**

* If the car has already payed, throw an Error:

**"{carNumber}'s driver has already payed his ticket."**

* Otherwise, this function set payed to true on the found car and **return:**

**"{carNumber}'s driver successfully payed for his stay."**

**getStatistics(carNumber)**

This **method** can be called **with one parameter** or **without** any.

If **NO** parameter is provided, the method should **return** the full information of the parking lot.

* At the first line:

**"The Parking Lot has { emptySlots } empty spots left."**

* On the lines, display information about each vehicle**, sorted alphabetically ascending** by their **carModel:**

**"{carModel} == {carNumber} - {Has payed / Not payed}"**

If the method is called with **parameter** for **carNumber**:

* **return only** the **information about the car with the given carNumber:**

**"{carModel} == {carNumber} - {Has payed / Not payed}"**

**Examples**

|  |
| --- |
| **Sample code usage** |
| **const parking = new Parking(12);**  **console.log(parking.addCar("Volvo t600", "TX3691CA"));**  **console.log(parking.getStatistics());**  **console.log(parking.pay("TX3691CA"));**  **console.log(parking.removeCar("TX3691CA"));** |
| **Corresponding output** |
| **The Volvo t600, with a registration number TX3691CA, parked.**  **The Parking Lot has 11 empty spots left.**  **Volvo t600 == TX3691CA - Not payed**  **TX3691CA's driver successfully payed for his stay.**  **TX3691CA left the parking lot.** |

1. class Parking {
2. constructor(capacity, vehicles = []) {
3. this.capacity = Number(capacity);
4. this.vehicles = vehicles;
5. }
7. addCar(carModel, carNumber) {
8. if (this.vehicles.length >= this.capacity) {
9. throw new Error("Not enough parking space.");
10. }
12. this.vehicles.push({ carModel, carNumber, payed: false })
13. return `The ${carModel}, with a registration number ${carNumber}, parked.`;
14. }
16. removeCar(carNumber) {
17. for (const curCar of this.vehicles) {
18. if (curCar.carNumber == carNumber) {
19. if (curCar.payed) {
20. this.vehicles = this.vehicles.filter(obj => obj.carNumber != carNumber);
21. return `${carNumber} left the parking lot.`
22. }
24. throw new Error(`${carNumber} needs to pay before leaving the parking lot.`);
25. }
26. }
28. throw new Error(`The car, you're looking for, is not found.`);
29. }
31. pay(carNumber) {
32. for (const curCar of this.vehicles) {
33. if (curCar.carNumber == carNumber) {
34. if (curCar.payed) {
35. throw new Error(`${carNumber}'s driver has already payed his ticket.`);
36. }
38. curCar.payed = true;
39. return `${carNumber}'s driver successfully payed for his stay.`
40. }
41. }
43. throw new Error (`${carNumber} is not in the parking lot.`)
44. }
46. getStatistics(carNumber) {
47. let output = '';
48. if (carNumber == undefined) {
49. output += `The Parking Lot has ${this.capacity - this.vehicles.length} empty spots left.\n`;
50. this.vehicles
51. .sort((a, b) => a.carModel.localeCompare(b.carModel))
52. .forEach(curObj => {
53. let isPayed;
54. curObj.payed ? isPayed = `Has payed` : isPayed = "Not payed";
55. output += `${curObj.carModel} == ${curObj.carNumber} - ${isPayed}\n`
56. })
57. return output.trim();
58. }

61. for (const curCar of this.vehicles) {
62. if (curCar.carNumber == carNumber) {
63. let isPayed;
64. curCar.payed ? isPayed = "Has payed" : isPayed = "Not payed";
66. return `${curCar.carModel} == ${curCar.carNumber} - ${isPayed}`;
67. }
68. }
69. }
70. }