# 03. Parking



*Parking games are also among the popular games. Let's create one.*

## Preparation

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

**Pay attention to name the package parking, 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.

**Car**

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

* **manufacturer: String**
* **model: String**
* **year: int**

The class **constructor** should receive the **manufacturer, model,** and **year**.You need to create the appropriate **getters and setters**. Override the **toString()** method in the following format:

**"{manufacturer} {model} ({year})"**

**Parking**

**Next**, write a Java class **Parking** that has **data** (**Collection**, which stores the entity **Car**). All entities inside the repository have the **same fields**. Also, the Parking class should have those fields:

* **type: String**
* **capacity: int**

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

* Field **data** – **Collection** that holds added cars
* Method add(Car car) – **adds** an **entity** to the data **if** **there** **is** an **empty cell** for the car.
* Method remove(String manufacturer, String model) – removes the car by **given manufacturer and model,** if such **exists**, and **returns boolean**.
* Method getLatestCar() – returns the **latest** car (by year) or null if have no cars.
* Method **getCar(String manufacturer, String model)** – returns the car with the **given manufacturer** and **model** or **null** if there is no such car.
* Getter getCount() – **returns** the **number** of cars.
* **getStatistics()** – **returns** a **String** in the following **format**:
  + **"The cars are parked in {parking type}:  
    {Car1}  
    {Car2}  
    (…)**"

## Constraints

* The **combinations** of **manufacturers** and **models** will be **always unique**.
* The **year** of the cars will always be **positive**.
* There won't be cars of the same year.

## Examples

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

|  |
| --- |
| Sample code usage |
| // Initialize the repository  Parking parking = new Parking("Underground parking garage", 5);  // Initialize entity  Car volvo = new Car("Volvo", "XC70", 2010);  // Print Car  System.out.println(volvo); // Volvo XC70 (2010)  // Add Car  parking.add(volvo);  // Remove Car  System.out.println(parking.remove("Volvo", "XC90")); // false  System.out.println(parking.remove("Volvo", "XC70")); // true  Car peugeot = new Car("Peugeot", "307", 2011);  Car audi = new Car("Audi", "S4", 2005);  parking.add(peugeot);  parking.add(audi);  // Get Latest Car  Car latestCar = parking.getLatestCar();  System.out.println(latestCar); // Peugeot 307 (2011)  // Get Car  Car audiS4 = parking.getCar("Audi", "S4");  System.out.println(audiS4); // Audi S4 (2005)  // Count  System.out.println(parking.getCount()); // 2  // Get Statistics  System.out.println(parking.getStatistics());  // The cars are parked in Underground parking garage:  // Peugeot 307 (2011)  // Audi S4 (2005) |

## Submission

Submit a **single .zip file**, containing the **parking package, with the classes inside (Car, Parking, 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.