# 03. Data Center



*Take a journey into the heart of modern technology infrastructure by developing a system to manage data centers. In this challenge, we will create a virtual repository that mirrors the organization and management of servers within a data center, focusing on their storage capacity and power usage. This task will test your ability to design and implement classes that encapsulate the properties and behaviors of data center components*

## Preparation

Download the provided skeleton in Judge. **Do not** change the **StartUp** class or its **namespace**.

Ensure that the project is named **DataCenter**, and all classes, fields, and methods should be named exactly as presented in the document. Maintain the project structure as described below.

## Problem description

Your task is to **create a repository** that stores **data center components** by implementing the following classes.

### Server

First, create a class **Server** with the following properties:

* **SerialNumber - string**
* **Model - string**
* **Capacity - int**
* **PowerUsage - int**

The class **constructor** should receive **serialNumber**, **model, capacity** and **powerUsage**.

Override the **ToString()** method in the following format:  
**"Server {SerialNumber}: {Model}, {Capacity}TB, {PowerUsage}W"**

### Rack

**Next**, create a class **Rack** that has **Servers** (a collection for storing components). All entities inside the repository have the **same properties**. The **Rack** class should have the following **properties**:

* **Slots – int** (representing the **maximum number of devices** it can hold)
* **Servers - List<Server>**
* **GetCount** – getter that **returns** the **number** of servers in the rack**.**

The class **constructor** should receive **slots.** Also, it should initialise the **Servers** with a **new instance** of the collection.

#### Implement the following features as class methods:

* **AddServer(Server server)** – **adds** a **Server** to the **Servers** **collection**, **if** there are **Slots** **available**. If a **Server** with the **same SerialNumber is already added**, do not duplicate servers, just **skip the command.**
* **RemoveServer(string serialNumber)** – attempts to find a **Server**, within the **Servers** collection,with **SerialNumber** value, **matching the given parameter**.
  + If **no sever is found**, the method returns boolean value - **False**
  + **Otherwise,** the server is **removed from the collection** and the method returns boolean value - **True**
* **GetHighestPowerUsage()** – **returns the ToString()** valueofthe **component with the greatest PowerUsage** value**,** among the **Servers** collection**.**
* **GetTotalCapacity() – returns** the sum of the **Capacity of all servers added to the collection.**
* **DeviceManager()** – **returns** a **string** **format** listing all operating servers:
  + **"{count} servers operating:  
    {server1}  
    {server2}  
    {…}**

**{servern}"**

[*See the Examples at the End of the Document*](#Report)

## Constraints

* Servers are always added, before commands manipulating the **Servers** collection are received.
* There will always be exactly one **Server** that **has the greatest value** of **PowerUsage** property.

## Examples

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

|  |
| --- |
| **Sample code usage** |
| //Initialize new repository (Rack) with slots for 8 servers  Rack rack = new(8);  //Initialize 10 servers  Server server1 = new("SN001", "Dell PowerEdge T340", 100, 450);  Server server2 = new("SN002", "HP Proliant DL360", 200, 220);  Server server3 = new("SN003", "Dell PowerEdge T340", 250, 350);  Server server4 = new("SN004", "IBM Power System S922", 220, 330);  Server server5 = new("SN005", "Lenovo ThinkSystem SR650", 250, 550);  Server server6 = new("SN006", "HPE Synergy 480 Gen10", 80, 180);  Server server7 = new("SN007", "Fujitsu PRIMERGY RX2530 M5", 120, 250);  Server server8 = new("SN008", "Dell EMC PowerEdge R740xd", 160, 380);  Server server9 = new("SN006", "Supermicro SuperServer 1029P-WTR", 150, 280);  Server server10 = new("SN009", "Cisco UCS B200 M5", 180, 400);  //Add servers to the rack, all servers should be added successfully  rack.AddServer(server1);  rack.AddServer(server2);  rack.AddServer(server3);  rack.AddServer(server4);  rack.AddServer(server5);  rack.AddServer(server6);  rack.AddServer(server7);  //Try to add a server with a duplicated SerialNumber to the rack  rack.AddServer(server9);  //Add server to the last available Slot  rack.AddServer(server8);  //Try to add a server when all slots are busy  rack.AddServer(server10);  //Remove a server with a valid SerialNumber, should return True  rack.RemoveServer("SN001");  //Remove a server with an invalid SerialNumber, should return False  rack.RemoveServer("SN011");  //Check servers count  Console.WriteLine(rack.GetCount);  //7  //Try to add a server when 1 slot is already empty  rack.AddServer(server10);  //Check servers count  Console.WriteLine(rack.GetCount);  //8  //Get the server with the highest power usage  Console.WriteLine(rack.GetHighestPowerUsage());  //Server SN005: Lenovo ThinkSystem SR650, 250TB, 550W  //Get the total capacity of the system  Console.WriteLine(rack.GetTotalCapacity());  //1460  //DeviceManager report string  Console.WriteLine(rack.DeviceManager());  //8 servers operating:  //Server SN002: HP Proliant DL360, 200TB, 220W  //Server SN003: Dell PowerEdge T340, 250TB, 350W  //Server SN004: IBM Power System S922, 220TB, 330W  //Server SN005: Lenovo ThinkSystem SR650, 250TB, 550W  //Server SN006: HPE Synergy 480 Gen10, 80TB, 180W  //Server SN007: Fujitsu PRIMERGY RX2530 M5, 120TB, 250W  //Server SN008: Dell EMC PowerEdge R740xd, 160TB, 380W  //Server SN009: Cisco UCS B200 M5, 180TB, 400W |

## Submission

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