**Rabbits**



* **Preparation**

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

* **Problem description**

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

First, write a C# class **Rabbit** with the following properties:

* **Name: string**
* **Species: string**
* **Available: bool - true by default**

The class **constructor** should receive **name and species**. Override the **ToString()** method in the following format:

**"Rabbit ({species}): {name}"**

**Next**, write a C# class **Cage** that has **data** (a collection which stores the entity **Rabbit**). All entities inside the repository have the **same properties**. Also, the **Cage** class should have those **properties**:

* **Name: string**
* **Capacity: int**

The class **constructor** should receive **name** 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 rabbits
* Method **Add(Rabbit rabbit)** - **adds** an **entity** to the data **if** **there** **is** **room** for it
* Method **RemoveRabbit(string name)** - removes a rabbit by **given name,** if such **exists**, and **returns bool**
* Method **RemoveSpecies(string species)** - removes **all rabbits** by given **species**
* Method **SellRabbit(string name)** - **sell** (**set** its **Available** **property** to **false** without removing it from the collection) the **first rabbit** with the **given name**, also **return** the **rabbit**
* Method **SellRabbitsBySpecies(string species)** - sells (**set** their **Available** **property** to **false** without removing them from the collection) and returns **all rabbits** from that **species as an array**
* Getter **Count** - **returns** the **number** of rabbits
* **Report()** - **returns** a **string** in the following **format, including only not sold rabbits**:
* **"Rabbits available at {cageName}:  
  {Rabbit1}  
  {Rabbit2}  
  (…)**"
* **Constraints**
* The **names** of the rabbits will be **always unique**.
* You will always have a rabbit added before receiving methods manipulating the Cage’s rabbits.
* **Examples**

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

|  |
| --- |
| **Sample code usage** |
| //Initialize the repository (Cage)  Cage cage = new Cage("Wildness", 20);  //Initialize entity  Rabbit rabbit = new Rabbit("Fluffy", "Blanc de Hotot");  //Print Rabbit  Console.WriteLine(rabbit); //Rabbit (Blanc de Hotot): Fluffy  //Add Rabbit  cage.Add(rabbit);  Console.WriteLine(cage.Count); //1  //Remove Rabbit  cage.RemoveRabbit("Rabbit Name"); //false  Rabbit secondRabbit = new Rabbit("Bunny", "Brazilian");  Rabbit thirdRabbit = new Rabbit("Jumpy", "Cashmere Lop");  Rabbit fourthRabbit = new Rabbit("Puffy", "Cashmere Lop");  Rabbit fifthRabbit = new Rabbit("Marlin", "Brazilian");  //Add Rabbits  cage.Add(secondRabbit);  cage.Add(thirdRabbit);  cage.Add(fourthRabbit);  cage.Add(fifthRabbit);  //Sell Rabbit by name  Console.WriteLine(cage.SellRabbit("Bunny")); //Rabbit (Brazilian): Bunny  //Sell Rabbit by species  Rabbit[] soldSpecies = cage.SellRabbitsBySpecies("Cashmere Lop");  Console.WriteLine(string.Join(", ", soldSpecies.Select(f => f.Name))); //Jumpy, Puffy  Console.WriteLine(cage.Report());  //Rabbits available at Wildness:  //Rabbit (Blanc de Hotot): Fluffy  //Rabbit (Brazilian): Marlin |

* **Submission**

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