# Price03. Groceries



*You will have to create a simple groceries management application by implementing the classes needed and the programming logic in the methods.*

**Preparation**

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

**Ensure that the project is named GroceriesManagement, 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 groceries repository** by implementing the following classes.

### Product

You are given a class **Product** with the following properties:

* **Name – string**
* **Price - double**

The class **constructor** should receive **name** and **price**.

* Override the **ToString()** method in the following format:  
  **"{Name}: {Price:F2}$"**

### GroceriesStore

**Next**, you have a **Groceries** class that contains **Stall** (a collection for storing products). All entries inside the repository have the **same properties**. The **GroceriesStore** class should have the following **properties**:

* **Capacity – int**
* **Turnover - double**
* **Stall – List<Product>**

The class **constructor** should receive **capacity**, to set the **Turnover's** initial value to **Zero** andinitialize **Stall** with new instance of the collection.

Implement the following features:

* **Method AddProduct(Product product)** – **adds** a **product** to the **Stall** **collection**, **if** the **Capacity** **allows it**. If there is a **Product** with the **same name already added**, do not duplicate products, just **skip the command.**
* **Method RemoveProduct(string name)** – attempts to find a **product by the given name** within the store's stall. If the **product is found**, it is removed from the stall and the method returns **true**. If the product with the specified name **does not exist** in the stall, the method returns **false**.
* **Method SellProduct(string name, double quantity)** – increases the **Turnover** by the **product's price multiplied by the quantity, rounded to the second decimal place** and **returns a string** in the format **{ProductName} - {TotalPrice:F2}$**. If the product is not found, return **"Product not found"**.
* **Method GetMostExpensive()** – **returns** the **ToString** value of the **most expensive** **Product** from the **Stall.**
* **Method CashReport() -** returns a formatted string that includes the **Turnover** property, which represents the **total amount of money** the store has made from selling products: **"Total Turnover: {Turnover:F2}$"**
* **Method PriceList()** – **returns** a **string** in the following **format**:
  + **"Groceries Price List:  
    {Product1}  
    {Product2}  
    {…}**

**{Productn}"**

### Constraints

* You will always have **products added** **before** receiving commands **manipulating** the collections in the GroceriesStore.
* The **Capacity** property is related to the **Stall** only.

### Examples

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

|  |
| --- |
| **Sample code usage** |
| //Initialize new repository (GroceriesStore)  GroceriesStore store = new(5);  //Initialize entities (Product)  Product apples = new("Apples", 1.20);  Product oranges = new("Oranges", 2.80);  Product bananas = new("Bananas", 1.50);  Product grapes = new("Grapes", 2.20);  Product watermelon = new("Watermelon", 1.90);  //Add products to the store  store.AddProduct(apples);  store.AddProduct(oranges);  store.AddProduct(bananas);  store.AddProduct(grapes);  store.AddProduct(watermelon);  //Attempt to add a sixth product which should be skipped due to capacity  Product cherries = new("Cherries", 5.70);  store.AddProduct(cherries);  //Remove existing Product  Console.WriteLine(store.RemoveProduct("Grapes")); //True  //Try to delete not existing Product  Console.WriteLine(store.RemoveProduct("Pears")); //False  //Try to add once again, if there is enough Capacity  store.AddProduct(cherries);  //Sell some products  Console.WriteLine(store.SellProduct("Apples", 1.5)); //Apples = 1.80$  Console.WriteLine(store.SellProduct("Bananas", 2.4)); //Banans = 3.60$  Console.WriteLine(store.SellProduct("Grapes", 2)); //Product not found  Console.WriteLine(store.SellProduct("Apples", 2.5)); //Apples = 3.00$  Console.WriteLine(store.SellProduct("Watermelon", 15)); //Watermelon = 28.50$  Console.WriteLine(store.SellProduct("Cherries", 0.5)); //Cherries = 2.85$  //Get the most expensive product  Console.WriteLine(store.GetMostExpensive()); //Cherries: 5.70$  // Generate and display a cash report  Console.WriteLine(store.CashReport());  // "Total Turnover: 39.75$"  //Display the price list  Console.WriteLine(store.PriceList());  // "Groceries Price List:  // Apples: 1.20$  // Oranges: 2.80$  // Bananas: 1.50$  // Watermelon: 1.90$  // Cherries: 5.70$" |

**Submission**

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