# Shopping Center – Sample Exam Problem

This document describes a **sample exam problem** for the ["Data Structures" course @ Software University](https://softuni.bg/trainings/1147/Data-Structures-June-2015) along with a step-by-step guidelines how to solve it.

## Problem Description

A **shopping center** keeps a set of **products**. Each product has **name**, **price** and **producer**. Your task is to model the shopping center and design a **data structure holding the products**. Write a program that executes **N** commands, given in the input (a single command at a line):

* AddProduct name;price;producer– adds a product by given name, price and producer. If a product with the same name / producer/ price already exists, the newly added product does not affect the existing ones (duplicates are allowed). As a result the command prints “**Product added**”.
* DeleteProducts producer– deletes all products matching given producer. As a result the command prints “**X products deleted**” where **X** is the number of deleted products or “**No products found**” if no such products exist.
* DeleteProducts name;producer – deletes all products matching given product name and producer. As a result the command prints “**X products deleted**” where **X** is the number of deleted products or “**No products found**” if no such products exist.
* FindProductsByName name – finds all products by given product name. As a result the command prints a list of products in format **{name;producer;price}**, ordered by name, producer and price. Print each product on a separate line. If no products exist with the specified name, the command prints “**No products found**”.
* FindProductsByProducer producer– finds all products by given producer. As a result the command prints a list of products in format **{name;producer;price}**, ordered by name, producer and price. You should print each product on a single line**.** If no products exist by the specified producer, the command prints “**No products found**”.
* FindProductsByPriceRange fromPrice;toPrice – finds all products whose price is greater or equal than **fromPrice** and less or equal than **toPrice**. As a result the command prints a list of products in format **{name;producer;price}**, ordered by name, producer and price. You should print each product on a separate line. If no products exist within the specified price range, the command prints “**No products found**”.

All string matching operations are **case-sensetive**.

### Input

The input data should be read from the console.

* At the first line you will be given the number **N** of the commands.
* At each of the next **N** lines you will be given a command in the format described above.

The input data will always be valid and in the described format. There is no need to check it explicitly.

### Output

The output data should be printed on the console.

The output should contain the output from each command from the input.

### Constraints

* **N** will be between 1 and 50 000, inclusive.
* All strings specified in the commands (e.g. product names and producers) consist of alphabetical characters, numbers and spaces. Strings are case-sensitive.
* Prices are given as real numbers with up to 2 digits after the decimal point, (e.g. 133.58, 320.3, or 10)
* The ‘.’ symbol is used as decimal separator.
* Prices should be printed with exactly **2 digits** after the decimal point (e.g. 320.30 instead of 320.3).
* Allowed working time for your program: **1.00 seconds** (at the judge environment).
* Allowed memory: **32 MB**.

### Examples

|  |  |
| --- | --- |
| **Input Example** | **Output Example** |
| 17  AddProduct IdeaPad Z560;1536.50;Lenovo  AddProduct ThinkPad T410;3000;Lenovo  AddProduct VAIO Z13;4099.99;Sony  AddProduct CLS 63 AMG;200000;Mercedes  FindProductsByName CLS 63 AMG  FindProductsByName CLS 63  FindProductsByName cls 63 amg  AddProduct 320i;10000;BMW  FindProductsByName 320i  AddProduct G560;999;Lenovo  FindProductsByProducer Lenovo  DeleteProducts Lenovo  FindProductsByProducer Lenovo  FindProductsByPriceRange 100000;200000  DeleteProducts Beer;Ariana  DeleteProducts CLS 63 AMG;Mercedes  FindProductsByName CLS 63 AMG | Product added  Product added  Product added  Product added  {CLS 63 AMG;Mercedes;200000.00}  No products found  No products found  Product added  {320i;BMW;10000.00}  Product added  {G560;Lenovo;999.00}  {IdeaPad Z560;Lenovo;1536.50}  {ThinkPad T410;Lenovo;3000.00}  3 products deleted  No products found  {CLS 63 AMG;Mercedes;200000.00}  No products found  1 products deleted  No products found |

### Submissions

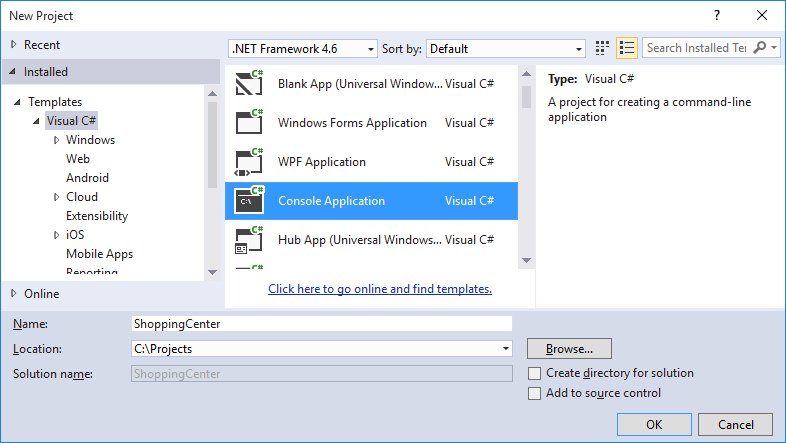
Submissions are accepted for automatic evaluation at the SoftUni judge system: <https://judge.softuni.bg/Contests/106/Data-Structures-Sample-Exam>.

## Step-by-Step Guidelines

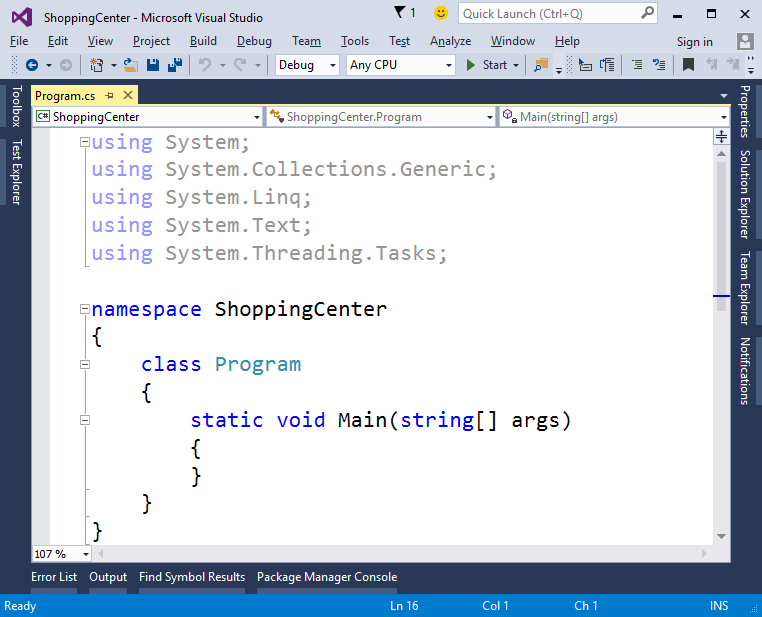
This section provides **step-by-step guidelines** for solving the “Shopping Center” problem in C#. You are invited to try to solve the above problem yourself and use these guidelines in case you have difficulties.

### Create a new C# Project in Visual Studio

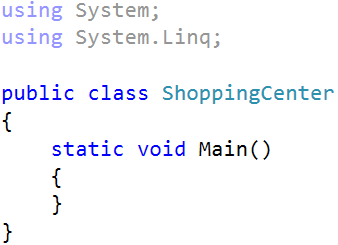
Create a new console application in Visual Studio called “ShoppingCenter”. Use [File] 🡪 [New] 🡪 [C#] 🡪 [Console Application]:



Visual Studio will create an empty C# project for you:



Remove the unneeded code (namespace, unneeded imports, etc.):



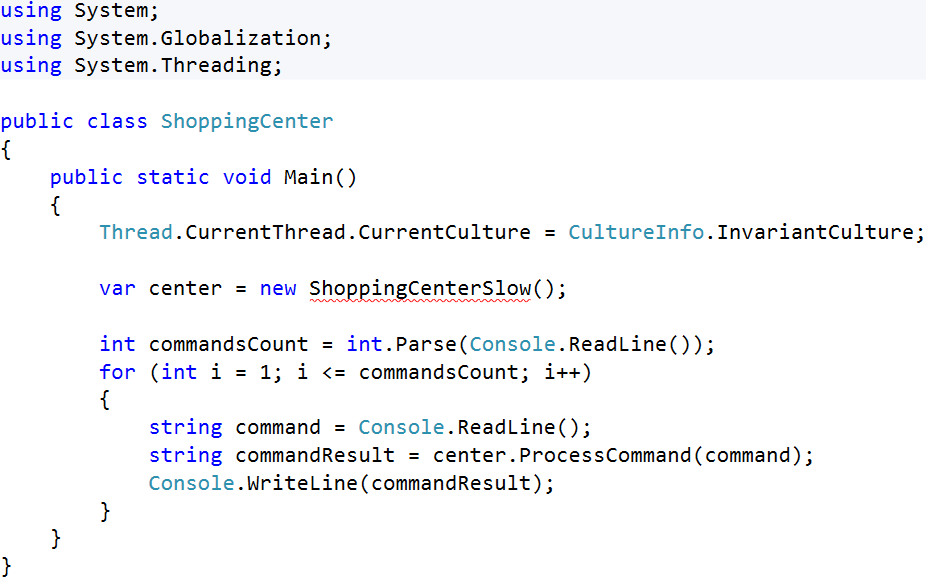
Rename the main (Program.cs) class to “ShoppingCenter.cs”:

|  |  |  |
| --- | --- | --- |
|  | 🡪 |  |

### Read the Input Data

Write some code to **read the input data** line by line and process each of the lines:

1. Reset the current culture to ensure that the decimal separator is “.” and the numbers will be displayed as usually (even when the current culture is Chinese, Japanese or Russian).
2. Instantiate the class ShoppingCenterSlow (it will be defined later).
3. Read the number of commands N.
4. Read the **next N lines** in a loop and process them.



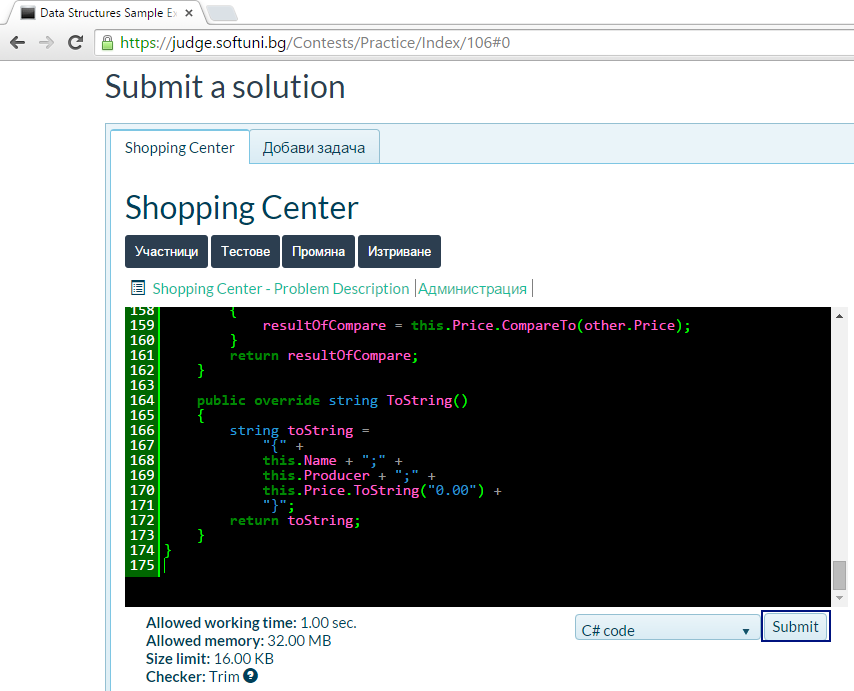
### List-Based Slow Implementation

Let’s start with a naive solution – the class ShoppingCenterSlow. It will implement the “shopping center” data structure based internally on List<Product> and will implement the commands by **queries with lambda expressions** in C# using LINQ:

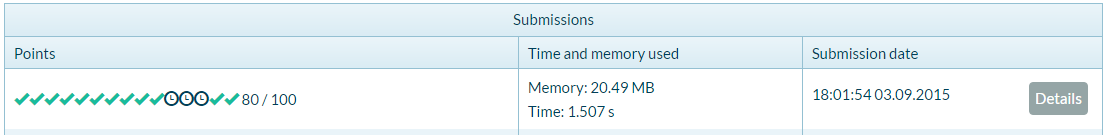
|  |
| --- |
| using System;  using System.Linq;  using System.Collections.Generic;  using System.Globalization;  using System.Threading;  using System.Text;  public class ShoppingCenter  {  public static void Main()  {  Thread.CurrentThread.CurrentCulture = CultureInfo.InvariantCulture;  var center = new ShoppingCenterSlow();  int commandsCount = int.Parse(Console.ReadLine());  for (int i = 1; i <= commandsCount; i++)  {  string command = Console.ReadLine();  string commandResult = center.ProcessCommand(command);  Console.WriteLine(commandResult);  }  }  }  public class ShoppingCenterSlow  {  private const string PRODUCT\_ADDED = "Product added";  private const string X\_PRODUCTS\_DELETED = " products deleted";  private const string NO\_PRODUCTS\_FOUND = "No products found";  private const string INCORRECT\_COMMAND = "Incorrect command";  private readonly List<Product> products = new List<Product>();  private string AddProduct(string name, string price, string producer)  {  Product product = new Product()  {  Name = name,  Price = decimal.Parse(price),  Producer = producer  };  this.products.Add(product);  return PRODUCT\_ADDED;  }  private string FindProductsByName(string name)  {  var products = this.products  .Where(p => p.Name == name)  .OrderBy(p => p);  return PrintProducts(products);  }  private string FindProductsByProducer(string producer)  {  var products = this.products  .Where(p => p.Producer == producer)  .OrderBy(p => p);  return PrintProducts(products);  }  private string FindProductsByPriceRange(string from, string to)  {  decimal rangeStart = decimal.Parse(from);  decimal rangeEnd = decimal.Parse(to);  var products = this.products  .Where(p => p.Price >= rangeStart && p.Price <= rangeEnd)  .OrderBy(p => p);  return PrintProducts(products);  }  private string PrintProducts(IEnumerable<Product> products)  {  if (products.Any())  {  var builder = new StringBuilder();  foreach (var product in products)  {  builder.AppendLine(product.ToString());  }  string formattedProducts = builder.ToString().TrimEnd();  return formattedProducts;  }  return NO\_PRODUCTS\_FOUND;  }  private string DeleteProductsByNameAndProducer(string name, string producer)  {  int countOfRemovedProducts =  this.products.RemoveAll(p => p.Name == name && p.Producer == producer);  if (countOfRemovedProducts > 0)  {  return countOfRemovedProducts + X\_PRODUCTS\_DELETED;  }  return NO\_PRODUCTS\_FOUND;  }  private string DeleteProductsByProducer(string producer)  {  int countOfRemovedProducts =  this.products.RemoveAll(p => p.Producer == producer);  if (countOfRemovedProducts > 0)  {  return countOfRemovedProducts + X\_PRODUCTS\_DELETED;  }  return NO\_PRODUCTS\_FOUND;  }  public string ProcessCommand(string command)  {  int indexOfFirstSpace = command.IndexOf(' ');  string method = command.Substring(0, indexOfFirstSpace);  string parameterValues = command.Substring(indexOfFirstSpace + 1);  string[] parameters =  parameterValues.Split(new char[] { ';' }, StringSplitOptions.RemoveEmptyEntries);  switch (method)  {  case "AddProduct":  return AddProduct(parameters[0], parameters[1], parameters[2]);  case "DeleteProducts":  if (parameters.Length == 1)  {  return DeleteProductsByProducer(parameters[0]);  }  else  {  return DeleteProductsByNameAndProducer(parameters[0], parameters[1]);  }  case "FindProductsByName":  return FindProductsByName(parameters[0]);  case "FindProductsByPriceRange":  return FindProductsByPriceRange(parameters[0], parameters[1]);  case "FindProductsByProducer":  return FindProductsByProducer(parameters[0]);  default:  return INCORRECT\_COMMAND;  }  }  }  public class Product : IComparable<Product>  {  public string Name { get; set; }  public decimal Price { get; set; }  public string Producer { get; set; }  public int CompareTo(Product other)  {  int resultOfCompare = this.Name.CompareTo(other.Name);  if (resultOfCompare == 0)  {  resultOfCompare = this.Producer.CompareTo(other.Producer);  }  if (resultOfCompare == 0)  {  resultOfCompare = this.Price.CompareTo(other.Price);  }  return resultOfCompare;  }  public override string ToString()  {  string toString =  "{" +  this.Name + ";" +  this.Producer + ";" +  this.Price.ToString("0.00") +  "}";  return toString;  }  } |

### Submit the List-Based Implementation to the Judge

Now, let’s **test** the slow list-based implementation in the SoftUni judge system:



It works pretty well, but some tests do not pass due to timeout:



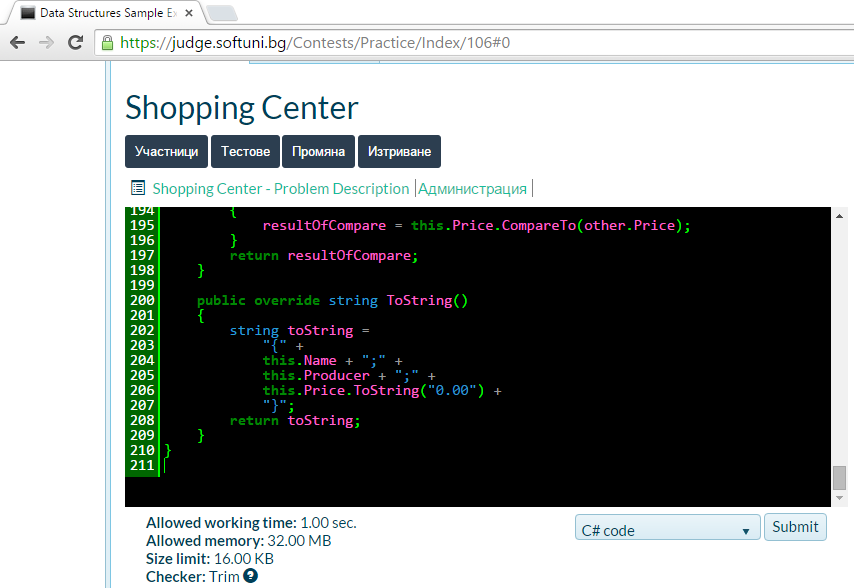
### Dictionary-Based Fast Implementation

Let’s start with a naive solution – the class ShoppingCenterFast. It will implement the “shopping center” data structure based internally on several **dictionaries from** WintellectPower Collections:

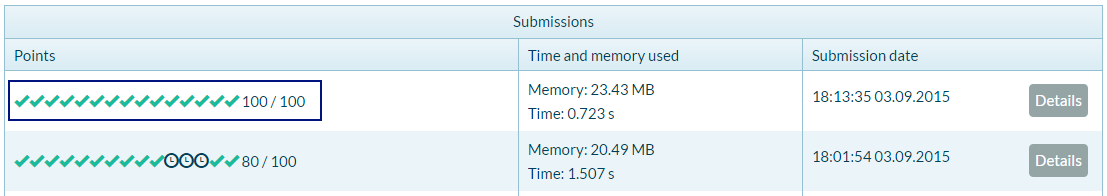
|  |
| --- |
| using System;  using System.Linq;  using System.Collections.Generic;  using System.Globalization;  using System.Threading;  using System.Text;  using Wintellect.PowerCollections;  public class ShoppingCenter  {  public static void Main()  {  Thread.CurrentThread.CurrentCulture = CultureInfo.InvariantCulture;  var center = new ShoppingCenterFast();  int commandsCount = int.Parse(Console.ReadLine());  for (int i = 1; i <= commandsCount; i++)  {  string command = Console.ReadLine();  string commandResult = center.ProcessCommand(command);  Console.WriteLine(commandResult);  }  }  }  public class ShoppingCenterFast  {  private const string PRODUCT\_ADDED = "Product added";  private const string X\_PRODUCTS\_DELETED = " products deleted";  private const string NO\_PRODUCTS\_FOUND = "No products found";  private const string INCORRECT\_COMMAND = "Incorrect command";  private readonly MultiDictionary<string, Product> productsByName =  new MultiDictionary<string, Product>(true);  private readonly MultiDictionary<string, Product> productsByNameAndProducer =  new MultiDictionary<string, Product>(true);  private readonly OrderedMultiDictionary<decimal, Product> productsByPrice =  new OrderedMultiDictionary<decimal, Product>(true);  private readonly MultiDictionary<string, Product> productsByProducer =  new MultiDictionary<string, Product>(true);  private string AddProduct(string name, string price, string producer)  {  Product product = new Product()  {  Name = name,  Price = decimal.Parse(price),  Producer = producer  };  this.productsByName.Add(name, product);  string nameAndProducerKey = this.CombineKeys(name, producer);  this.productsByNameAndProducer.Add(nameAndProducerKey, product);  this.productsByPrice.Add(product.Price, product);  this.productsByProducer.Add(producer, product);  return PRODUCT\_ADDED;  }  private string CombineKeys(string name, string producer)  {  string key = name + ";" + producer;  return key;  }  private string FindProductsByName(string name)  {  var productsFound = this.productsByName[name];  return SortAndPrintProducts(productsFound);  }  private string SortAndPrintProducts(IEnumerable<Product> products)  {  if (products.Any())  {  var sortedProducts = new List<Product>(products);  sortedProducts.Sort();  var builder = new StringBuilder();  foreach (var product in sortedProducts)  {  builder.AppendLine(product.ToString());  }    // Remove the undneeded last "new line"  builder.Length -= Environment.NewLine.Length;  string formattedProducts = builder.ToString();  return formattedProducts;  }  return NO\_PRODUCTS\_FOUND;  }  private string FindProductsByProducer(string producer)  {  var productsFound = this.productsByProducer[producer];  return SortAndPrintProducts(productsFound);  }  private string FindProductsByPriceRange(string from, string to)  {  decimal rangeStart = decimal.Parse(from);  decimal rangeEnd = decimal.Parse(to);  var productsFound = productsByPrice.Range(rangeStart, true, rangeEnd, true).Values;  return SortAndPrintProducts(productsFound);  }  private string DeleteProductsByNameAndProducer(string name, string producer)  {  string nameAndProducerKey = name + ";" + producer;  var productsToBeRemoved = productsByNameAndProducer[nameAndProducerKey];  if (productsToBeRemoved.Any())  {  int countOfRemovedProducts = productsToBeRemoved.Count;  foreach (var product in productsToBeRemoved)  {  productsByName.Remove(product.Name, product);  productsByProducer.Remove(product.Producer, product);  productsByPrice.Remove(product.Price, product);  }  productsByNameAndProducer.Remove(nameAndProducerKey);  return countOfRemovedProducts + X\_PRODUCTS\_DELETED;  }  return NO\_PRODUCTS\_FOUND;  }  private string DeleteProductsByProducer(string producer)  {  var productsToBeRemoved = productsByProducer[producer];  if (productsToBeRemoved.Any())  {  foreach (var product in productsToBeRemoved)  {  productsByName.Remove(product.Name, product);  string nameAndProducerKey = this.CombineKeys(product.Name, producer);  productsByNameAndProducer.Remove(nameAndProducerKey, product);  productsByPrice.Remove(product.Price, product);  }  int countOfRemovedProducts = productsByProducer[producer].Count;  productsByProducer.Remove(producer);  return countOfRemovedProducts + X\_PRODUCTS\_DELETED;  }  return NO\_PRODUCTS\_FOUND;  }  public string ProcessCommand(string command)  {  int indexOfFirstSpace = command.IndexOf(' ');  string method = command.Substring(0, indexOfFirstSpace);  string parameterValues = command.Substring(indexOfFirstSpace + 1);  string[] parameters =  parameterValues.Split(new char[] { ';' }, StringSplitOptions.RemoveEmptyEntries);  switch (method)  {  case "AddProduct":  return AddProduct(parameters[0], parameters[1], parameters[2]);  case "DeleteProducts":  if (parameters.Length == 1)  {  return DeleteProductsByProducer(parameters[0]);  }  else  {  return DeleteProductsByNameAndProducer(parameters[0], parameters[1]);  }  case "FindProductsByName":  return FindProductsByName(parameters[0]);  case "FindProductsByPriceRange":  return FindProductsByPriceRange(parameters[0], parameters[1]);  case "FindProductsByProducer":  return FindProductsByProducer(parameters[0]);  default:  return INCORRECT\_COMMAND;  }  }  }  public class Product : IComparable<Product>  {  public string Name { get; set; }  public decimal Price { get; set; }  public string Producer { get; set; }  public int CompareTo(Product other)  {  int resultOfCompare = this.Name.CompareTo(other.Name);  if (resultOfCompare == 0)  {  resultOfCompare = this.Producer.CompareTo(other.Producer);  }  if (resultOfCompare == 0)  {  resultOfCompare = this.Price.CompareTo(other.Price);  }  return resultOfCompare;  }  public override string ToString()  {  string toString =  "{" +  this.Name + ";" +  this.Producer + ";" +  this.Price.ToString("0.00") +  "}";  return toString;  }  } |

### Submit the List-Based Implementation to the Judge

Now, let’s **test** the fast dictionary-based implementation in the SoftUni judge system:



Now it runs correctly and fast enough:



**Congratulations!** You have solved the “shopping center” problem.