IDENTIFICATION: Brushup # 4 / PELE

Overall Purpose

The overall purpose for the group of ‘Brush-up’ assignments is to be able to quick, safe and sure in creating and using model classes, methods using collections and basic string operations.

The whole group of assignments consist of five steps:

1. A simple model class with constraints.
2. More advanced and complex model classes.
3. Testing a model class.
4. **Methods deal with collections. (This Assignment)**
5. String manipulations.

Background Material:

C# note by Per Laursen from 1st semester
(https://github.com/perl-easj/Teaching-materials/tree/master/CSharpProgramming/Notes)
Chapters: prog2.

This Assignment: Brushup # 4

Purpose
The purpose of this assignment is to search, collect, find from a collection by hand code, meaning do not use any LINQ buildin methods.

Mission
You are to

a. Expand the MenuCard, with several methods
b. Create a class with numbers

Domain description
In overall, you are to design and implement a methods and algorithms.

Assignment 1: Expand The MenuCard class

Exercise A:

In the MenuCard class add a method:

```
public Drink GetTheMostExpensiveDrink()
```

Returning the drink, which have the highest price.

Exercise B:

In the MenuCard class add a method:

```
public Drink GetTheCheapestDrink()
```

Returning the drink, which have the lowest price.

Exercise C:

In the MenuCard class add a method:

```
public Drink GetTheCheapestDrink(String TypeOfDrink)
```

Returning the drink, which have the lowest price of the specified type
(the type could be an enum if you have made one of the additional exercises).

Exercise D:

In the MenuCard class add a method:

```
public List<String> GetTypeOfDrinks()
```

Returning a list of all kind of type of drinks in the MenuCard
(the type could be a list of enum if you have made one of the additional exercises).

Exercise E:

In the MenuCard class add a method:

```
public List<Drink> GetTheCheapestDrinks()
```

Returning a list, with one drink for each type and have the lowest price.

Exercise F:

In the MenuCard class add a method:

```
public List<Drink> GetNonAlcoholicDrinks()
```

Returning a list of all drinks, that are non-alcoholic.

Exercise G:

In the MenuCard class add a method:

```
public double PriceOfCheapestDish()
```

Returning the price of the cheapest Dish.

Exercise G:

In the MenuCard class add a method:

```
public double PriceOfCheapestDish(String TypeOfDish)
```

Returning the price of the cheapest Dish of the specified.

Exercise H:

In the MenuCard class add a method:

```
public double GetAverageDishPrice()
```

Returning the average price of all dishes.

Exercise I:

In the MenuCard class add a method:

```
public List<Dish> GetDishOfType()
```

Returning a list of Dishes in order first all starters then all main and in the end desserts. (Hint in the result list first add all starters, then …. i.e. three loops)

Exercise J:

In the MenuCard class add a method:

```
public List<Dish> GetDishes(String typeOfDish, int number)
```

Returning a list of dishes all of the specified typeOfDish. Then number specify the number of dishes in the list. E.g. GetDishes(“main”, 3) gives a list of three main courses.

Exercise K:

In the MenuCard class add a method:

```
public List<Dish> GetCheapestDishes(String typeOfDish, int
number)
```

Returning a list of dishes all of the specified typeOfDish. Then number specify the number of dishes in the list but it should be the cheapest dishes. E.g. GetCheapestDishes(“main”, 3) gives a list of the three cheapest main courses.

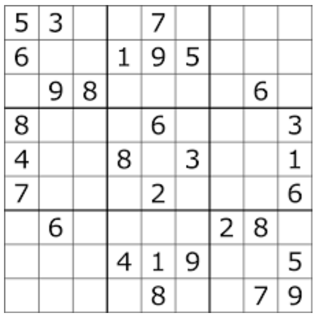
Assignment ADDITIONAL: for all exercises A-K, make a UnitTest.

Assignment 2: Create Sudoku class

<u>Step 1:</u>

Create a new class in your library – Sudoku class.
The idea is to start on making a Sudoku.

A Sudoku look like this:



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

The rules are shortly; each row and each column must contain the numbers 1-9, the same for each small 3*3 squares – for more detail see https://sudoku.com/how-to-play/sudoku-rules-for-complete-beginners/.

The class must have a 9 * 9 array of numbers (int or byte if you prefer).

i.e. the class have:

- An instance field of an double array of int. (not any list or …, only old fashion arrays)
- A property to get the array (no set the array)

IN THIS VERSION – you operate with a zero (0) is nothing, so you do not need to handle the blank.

- Constructor that initialise the whole array to zeros

<u>Step 2:</u>

Make a method Insert(int r, int c, int value), to insert the 'value' at r-row and c-column.

(Make a check that the value is in the range 1-9)

<u>Step 3:</u>

Make a method CheckRow(int r), to check if a r-row is full i.e. all 9 'cells' have a value, and they have the numbers 1 to 9.

<u>Step 4:</u>

Make a method CheckCol(int c), to check if a c-column is full i.e. all 9 'cells' have a value, and they have the numbers 1 to 9.

<u>Step 5:</u>

Make a method CheckSquare(int r,int c), to check if a square formed by rows r to (r+2) and the columns c to (c+2) is full i.e. all 9 'cells' have a value, and they have the numbers 1 to 9.