Homework. Do all the tasks in one VS solution.

NB! The goal of this homework is testing!

Useful materials (checking and doing example from first link is highly recommended):

- NUnit tutorial and example (use the orange buttons to view next pages also): http://dotnetpattern.com/nunit-introduction
- Long video about principles of unit testing and NUnit https://www.youtube.com/watch?v=HYrXogLj7vg
- Short video about NUnit setup: https://www.youtube.com/watch?v=f2NrKazjWes
- Nunit documentation and tutorial (menu is on the right): https://github.com/nunit/docs/wiki/NUnit-Documentation
- Writing good unit tests: https://esj.com/articles/2012/09/24/better-unit-testing.aspx
- Some info about unit testing in general: https://www.guru99.com/unit-testing-guide.html

Don't forget that you need to add NUnit to every project!

Exercise 1 (create all the tasks in exercise 1 in the same project):

Create a new class called Exercises and add all the methods there. Then create a second class called ExercisesTesting and add all the tests there. Each exercise should have at least 4 tests, if needed then also more.
In that class create:

1) Method for joining together two strings. The strings are given as a parameter and result string is returned.
   Example: JoinStrings(„a", „b") -> „ab"

2) Method for generating and returning an array of 6 numbers where:
   - one number is 100
   - 2 numbers are in a range from 0 to 20, 2 numbers from 30-60 and 2 numbers 100-200
   - one number divides by 5 (5,10,15,20,25 etc)

   Make (some numbers) of this array different every time; use Random for generating them.

   Test all the requirements as well as you can. Examples: http://nunit.org/docs/2.6/collectionConstraints.html

3) Method for calculating BMI. https://www.bmi3d.com/formula.html
   This method takes height and weight as parameters and returns BMI value as text. Example: *CalculateBMI(165.3, 62) -> „Normal"*

BMI	Weight Status
Below 18.5	Underweight
18.5 - 24.9	Normal
25.0 - 29.9	Overweight
30.0 and Above	Obese

Test all the 4 cases.

4) Create a method that takes string (a sentence) as input and puts * sign between words. Test cases where there are multiple spaces between words; expected output is always one * sign.

   Example:
   *ReplaceSpaces(„I am a dog“) -> „I*am*a*dog“*
   *ReplaceSpaces(„Nice Day“)-> „Nice*Day“*

There are many different ways this can be solved. One option is using char array and checking space character: https://www.dotnetperls.com/tochararray
Useful methods can be also found: https://www.dotnetperls.com/split, https://www.dotnetperls.com/string-join, https://www.dotnetperls.com/replace

Exercise 2:

Create a new class called TimeCalculator

Create a method for calculating time zone differences which takes the amount of hours to add or subtract and calculates the new time. Parameter should be double (we can also add 0.5 hours for example).

- Initial time value is 01.01.2000 00:00.
- Result should contain date and time (with hours and minutes; seconds are not needed).
- Seconds are not needed. Method parameter has either + or – sign to determine weather to add or subtract values.

Examples:

FindTime(+2) -> 01.01.2000 02:00

FindTime(-1) -> 31.12.1999 23:00

FindTime(+25.5) -> 02.01.2000 01:30

Divide this task into sub methods (addDay, subtractDay, findMinutesFromInput etc and test them separately). (Or get faimilar with DateTime class and use built-in methods).

Read: https://www.dotnetperls.com/datetime, http://nunit.org/docs/2.4.8/equalConstraint.html, http://nunit.org/docs/2.5/equalConstraint.html, https://stackoverflow.com/questions/3577856/nunit-assert-areequal-datetime-tolerances

Exercise 3

Look at the money and wallet example and try to understand it! There are few methods still missing, try to complete them.

Download this code and copy classes Wheel and Car from ained.ttu.ee https://ained.ttu.ee/mod/resource/view.php?id=7647 and add them to your project. (Or copy paste the code into your own solution from the end of this file).

Tasks:

- Add a method for comparing two wheel objects into Wheel class. Wheel to compare to is added as a parameter. Wheels are equal if their diameter and make are equal. Test this method (Try comparing 2 wheel objects). Method return type is bool.

  Example: wheel1.IsEqual(wheel2) //returns true or false

- Add a test to verify that car has 0 wheels after it is first created (and no methods are called).
- Add a test to verify that car has 4 wheels after `CreateFourWheels` method is called.
- Add field age for tires and 2 methods:
  - One for setting the value for age and other one for getting the value (returns the value for age field).

Default age is 0 years. Test the method for getting age.

Example: wheel1.SetAge(10) //age field value is now 10. GetAge() would return 10

- Add a method for class Car for checking the need to change tires. Method return type is bool.

You should replace tires if one (or more) of the conditions is true:

- Not all the tires are the same size (car has tires with different sizes)
- Tire age is more than 5
- Car has less (or more) than 4 wheels

Example:

- car has tires that all have size 16 and age 3: *ShouldChangeTires() -> false*
- car has 3 tires with size 14 and one tire with size 15: *ShouldChangeTires() -> true*
- car has one or more tires which have age more than 5: *ShouldChangeTires() -> true*
- car has 2 tires: *ShouldChangeTires() -> true*

Test this method at least 4 times for different results (positive, negative etc). Think of preconditions (and how to achieve them) for this test!

Code for classes Wheel and Car:

```
class Wheel
{
    private int diameter;
    private string make;
```

```
    //constructor, takes diameter and maker company as a parameter
    public Wheel(int diameterToSet, string makerName)
    {
        diameter = diameterToSet;
        make = makerName;
    }

    //method for changing the maker of the wheel
    public void ChangeMaker(string newMake)
    {
        make = newMake;
    }

    //method for printing info about a wheel
    public void PrintInfo()
    {
        Console.WriteLine("Wheel size is {0} and company {1}", diameter,  make);
    }

    //helper methods for making private properties visible
    public int GetDiameter()
    {
        return diameter;
    }

    public string GetMake()
    {
        return make;
    }
}

class Car
{
    List<Wheel> wheels = new List<Wheel>(); //List for storing wheel objects

    //creates 4 wheel objects and adds them to a list
    public void CreateFourWheels(int size, string makerName)
    {
        for (int i = 0; i < 4; i++)
        {
            wheels.Add(new Wheel(size, makerName));
        }
    }

    //returns wheel object from wheels list
    public Wheel GetTireByIndex(int index)
    {
        //avoid exception and check if index is present in the list
        if (index < wheels.Count)
        {
            return wheels[index]; //return the wheel object
        }
        else
        {
            Console.WriteLine("Car does not have wheel with this index!");
            return null; //return empty object; method always has to return something
        }
    }

    //calls PrintInfo() from all wheel objects in a list
    public void PrintWheelInfo()
    {
        Console.WriteLine();
        Console.WriteLine("This car has {0} wheels", wheels.Count);
        foreach (Wheel wheel in wheels)
        {
            wheel.PrintInfo();
        }
    }
```

```
}
```