C# Practice 2 – Create classes and objects

1. **Animal competition**.
   In our city there will be held an Animal Tournament. Help this event with a C# program.
   Registering an **animal**, it is necessary to give its *name* and *birth year*. These data can't be changed during competition. Each animal is *scored*, it gets points for its *beauty* and its *behaviour*. Determining the resulting *score*, the age of an animal is important (now age is only in years). Over a unified *age limit* their score will be 0, under this limit the score is computed. In the case of a young animal the beauty is more important, in the case of an older one the behaviour is significant. The formula is the next: score=age*behavior+(age limit-age)*beauty. (For example if the age limit is 10 years, then the score of a 2 years old animal is: 2*behavior + (10-2)*beauty.)

   If you want to print the data of an animal, please use the ToString() method.

   Please, give the value of **actual year** and **age limit**, and let the competition begin. This mean: while there is an animal, please register it (read on its data), and after it generate two random numbers – one for beauty and one for behavior – and let the "jury" score this animal. After it please print the data of animal.

   At the end of competition print out the number of animals, the sum of their score and the value of maximal score.
2. A businessman wants to operate **a vehicle fleet** and we are asked to create a C# application to maintain the data of vehicles and the information needs to operation. Vehicles have attributes associated with them that need to be tracked. It is important to store the chassis number, the registration number of the vehicle, the year of manufacture, and we can calculate its age. Important to store their fuel consumption as well. This value can change and we know that this value depends on the age of the vehicle and its mechanical state too. Along with the properties of the vehicles, our application also needs an established set of behaviors exposed by the Vehicle object.
   a.) Design the solution with UML diagram, write the code, then run the program!
   b.) Expand the knowledge of the program store how many kilometers have been running during the life of the vehicle.
   c.) Resolve that you can adjust the current petrol price of each vehicle are equally valid and remodel written in the previous section method so that the currently specified route length and the knowledge of 100 km per consumption calculates the expected costs.
3. Create a **football player** (footballer) class! Every football player has a *name* and *identification number*. Football players *train(),* and their *training time* is increased by the actual time in hours spent training. Every player *plays*() matches too. This means that the *number of playing* of the football player is increased by one. When the football player *kicks() a goal* the player's *score* is increased by one. Every football player has a *base salary*. If a player's training time is higher than a *unified training-limit value* then an extra wages depending of the training time over the limit will be add to the salary. This *hourly training rate* is unified for everybody. Football players can get extra wages for their playing also. Over a *unified score limit* their wages are increased by the number of playing multiply with a *unified playing fee* value.

Read the limit values. While there is a football player, read the data. After it generate random number for training hours and decide randomly that the player plays a match or not. Create a random number for score goal also.
At the end print the data of the football player using ToString() method and print the calculated wages also.

4. Create an **Employee** class. Items to include as data members are employee number, name, date of hire, and monthly salary. Include appropriate constructors and properties. Override the ToString ( ) method to return all data members. Create a second (control) class to test your Employee class.
5. Write a program that includes an **Employee** class that can be used to calculate and print the take-home pay for a commissioned sales employee. All employees receive 7% of the total sales. Federal tax rate is 18%. Retirement contribution is 10%. Social Security tax rate is 6%. Use appropriate constants, design an object-oriented solution, and write constructors. Create a control class to test your design. Allow the user to enter values for the name of the employee and the sales amount for the week in the control class.
6. Create a class named **Taxpayer**. Data fields for Taxpayer objects include the Social Security number (use a string for the type, but do not use dashes within the number), the yearly gross income, and the tax owed. Include a property with get and set accessors for the first two data fields, but make the tax owed a read-only property. The tax should be calculated whenever the income is set. Assume the tax is 15 % of income for incomes under $30,000 and 28 % for incomes that are $30,000 or higher. Write a program to test your class. Prompt the user for data and create 5 different objects and display them.
7. Create a class named **Car** with auto-implemented properties for the vehicle ID number, make, model, color, and value of a Car object. Write a DisplayFleet( ) method that accepts any number of Car objects, displays their values, and displays the total value of all Car objects passed to the method. Write a Main( ) method that declares five Car objects and assigns values to each, then calls DisplayFleet() three times—passing three, four, and five Car objects in successive calls.
8. Create an **Invoice** class that could be used by a bookstore. Items to include as data members are item number, description, unit price, and quantity purchased. Include appropriate constructors and properties plus an additional method that calculates the total cost using the quantity and unit price. Override the ToString ( ) method to return the item description and total cost. Create a control class to test your Invoice class.